

Self-Organized Robust Optimization in Open Technical Systems

*Self-Organization and Computational Trust for
Scalable and Robust Resource Allocation under Uncertainty*

Gerrit Anders

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Angewandte Informatik
der Universität Augsburg, 2016

Erstgutachter: Prof. Dr. Wolfgang Reif
Zweitgutachter: Prof. Dr. Jörg Hähner
Drittgutachter: Prof. Dr. Christian Müller-Schloer

Tag der mündlichen Prüfung: 25. April 2016

Summary

Resource allocation, in terms of balancing supply and demand, is a common problem in supply systems, such as electric power systems. Given that these systems are mission-critical – that is, their failure can have massive consequences for people, industries, and public services –, it is of the utmost importance that they maintain the balance under all circumstances. If the system components cannot arbitrarily change their supply for the sake of balance within a fixed period of time, resources have to be allocated in the form of schedules for a number of time steps in advance. In future power systems, maintaining the balance between supply and demand will become an extremely challenging optimization task. Such systems will be characterized by a vast number of distributed energy resources, including weather-dependent power plants and small dispatchable generators, as well as new types of consumers. A key aspect to deal with the complexity and the uncertainties in future power systems, is to enable the system components to act autonomously in their environment, to maintain efficient organizational structures, and to anticipate uncertainties originating from the behavior of the other components.

The result of this thesis is an integrated approach to robust resource allocation in open technical systems that is based on the principles of self-organization and computational trust. It introduces *Trust-Based Scenario Trees* as a trust model to quantify and anticipate uncertainties emanating from volatile demand that follows different behavioral patterns. Trust-Based Scenario Trees function as the basis for finding robust solutions to the scheduling problem, that is, the optimization problem of creating suitable schedules. Further, this thesis presents methods for *self-organizing hierarchical system structures* that serve as an approach to autonomous problem decomposition in large-scale open technical systems. These methods comprise *partitioning constraints*, *homogeneous partitioning* as an underlying organizational paradigm, and the two self-organization algorithms *PSOPP* and *SPADA*. While the partitioning constraints specify the shape of the hierarchy, homogeneous partitioning defines the desired composition of the subsystems residing in the hierarchy. The two self-organization algorithms PSOPP and SPADA enable the system components to maintain an adequate hierarchical structure that supports the system's goals. These methods lay the foundation for the system's robustness, efficiency, and scalability. Moreover, the thesis outlines concepts and optimization algorithms for *robust resource allocation* in self-organizing hierarchies. In detail, it specifies robust solutions to the scheduling problem that allow the system components to deal with different possible developments of the demand; created schedules rely on Trust-Based Scenario Trees. For the timely creation of high-quality robust solutions, the thesis presents the auction- and trust-based scheduling algorithm *TruCAOS* that reduces the complexity of the scheduling problem by enabling the components to actively participate in the process of schedule creation.

All concepts and algorithms devised in this thesis have been analyzed in extensive empirical evaluations in an elaborate simulation environment for autonomous power systems on the basis of real world data.

Contents

I	Self-Organized Robust Resource Allocation in Open Technical Systems	1
1	Introduction	3
1.1	The Resource Allocation Problem	4
1.2	The Scheduling Problem – Proactive Resource Allocation to Address the Inert Provision of Resources	6
1.3	Challenges of Resource Allocation in Open Technical Systems	7
1.4	Self-Organization, Computational Trust, and Robustness	9
2	Decentralized Power Management	13
2.1	Current and Future Power Systems	13
2.2	Autonomous Virtual Power Plants – a Trust-Based Multi-Agent Approach to Self-Organizing Autonomous Power Management	14
2.3	Smart Grid Projects	17
3	Contributions of this Thesis: Self-Organization and Computational Trust for Scalable and Robust Resource Allocation under Uncertainty	19
II	Computational Trust as Enabler for Self-Organized Robust Resource Allocation	23
4	Trust as a Measure of Uncertainty	25
4.1	Properties of Trust	26
4.2	Quantifying and Anticipating Uncertainties in Predicted Demand and Scheduled Supply by Means of Trust Values	28
5	Trust-Based Scenario Trees to Anticipate Volatile Demand that Follows Different Behavioral Patterns	33
5.1	Related Work	35
5.2	Trust-Based Scenario Trees	37
5.3	Evaluation in the Decentralized Power Management Case Study	41
III	Self-Organizing Hierarchical System Structures as Foundation of Robustness and Scalability	49
6	Self-Organizing Partitions for Autonomous Hierarchical Problem Decomposition	51
6.1	The Partitioning Problem	57

6.2	Partitioning Constraints for Guiding Self-Organized Hierarchical Problem Decomposition . . .	58
6.3	Homogeneous Partitioning to Promote System Stability and Robust Resource Allocation . . .	60
6.4	Initiating the Reorganization of a Partitioning and Adopting the Reorganization Result . . .	67
6.5	Related Work	69
7	PSOPP – a Particle Swarm Optimizer for Solving the Partitioning Problem	79
7.1	Particle Swarm Optimization	81
7.2	Basic Procedure of PSOPP	83
7.3	Similarity of Partitionings	84
7.4	Random Moves in the Search Space	85
7.5	Approach of Other Candidate Solutions	88
7.6	Evaluation and Comparison to Related Approaches	90
8	SPADA – a Decentralized Agent-Based Partitioning Algorithm	99
8.1	The Acquaintances Graph	101
8.2	Decentralized Formation of Partitions	103
8.3	Satisfaction of Partitioning Constraints	112
8.4	Applying Partition Changes to the Acquaintances Graph	117
8.5	Evaluation and Comparison to PSOPP	121
9	Self-Organized Formation of Hierarchical System Structures	135
9.1	The Hierarchical Control Loop	137
9.2	Evaluation in the Decentralized Power Management Case Study	140
IV	Robust Solutions to the Resource Allocation Problem in Self-Organizing Hierarchies	151
10	TruCAOS – an Auction- and Trust-Based Mechanism for Scalable Schedule Creation	153
10.1	Basic Procedure of TruCAOS	155
10.2	Identification of Winner Proposals	159
10.3	Trust Metrics for Quantifying and Anticipating Uncertainties in Promised Supply and Predicted Demand	162
10.4	Trust-Based Measures to Deal with Uncertain Supply and Demand	165
10.5	Evaluation in the Decentralized Power Management Case Study	167
11	Robust Solutions to the Scheduling Problem on the Basis of Trust-Based Scenario Trees	177
11.1	Formalization of the Scenario-Based Scheduling Problem	181
11.2	Top-Down Creation of Scenario-Based Schedules	184
11.3	Provision of Reserves in Hierarchical Systems	191
11.4	Robust and Cooperative Resource Allocation with TruCAOS	194
11.5	Evaluation in the Decentralized Power Management Case Study	195
12	Proactively Guided Reactive Supply Adjustments	201
12.1	Reactive Compensation for Deviations in Hierarchical Systems	203
12.2	Determining Proactively Guided Reactive Supply Adjustments	206
12.3	Evaluation in the Decentralized Power Management Case Study	210
13	Related Work	215

V Conclusion and Outlook	221
14 Summary of Research Contributions and Evaluation Results	223
15 Open Research Challenges and Future Directions	227
List of Figures	231
List of Tables	240
Bibliography	243
Own Peer-Reviewed Publications	261

Part I

Self-Organized Robust Resource Allocation in Open Technical Systems

Chapter 1 introduces the system class, the resource allocation problem to be solved, and the challenges of solving the problem in open technical systems by the example of future decentralized power systems. Chapter 2 presents our case study in detail, which illustrates our findings and serves to evaluate the devised concepts and algorithms. Chapter 3 outlines the contributions of this thesis and gives an insight into its structure.

Introduction

Summary. Introduces the system class, the resource allocation problem to be solved, and the challenges of solving the problem in open technical systems by the example of future decentralized power systems. Due to the power systems' mission-critical nature, their stability and availability is of utmost importance. Future decentralized power systems call for solutions that cope with a large number of system components and the uncertainties the system is exposed to.

Supply systems, such as electric power systems, gas pipeline systems, district heating systems, and water supply systems, influence our daily life and constitute the backbones of our society. Systems of this class have in common that their task is to solve a *resource allocation problem* [51]. That is, their goal is to stipulate the *supply* of the system components, i.e., the contribution of resources, in a way that their sum satisfies a given *demand* that is imposed by the environment or other components in the system. Neither surplus nor shortage is desirable and, often, even feasible without risking to damage the system's infrastructure. In gas pipeline and water supply systems, for instance, the challenge is to maintain the system's pressure at a certain level [62, 147]. Regarding district heating systems, the network temperature has to be kept between specific bounds [34]. Similarly, the main task in power systems is to maintain the balance between power production and consumption at all times [214]. While the satisfaction of the demand is paramount, the achieved balance should further be kept at low overall costs.

Supply systems are inherently mission-critical. Because their failure can have massive consequences for people, industries, and public services, the system's stability and availability is of utmost importance. This is a particularly challenging task because the class of supply systems is a representative of *open systems*, that is, we can make only few assumptions about the system's scale, the behavior of participants, and its environment [22, 87]. In such systems, the benevolence assumption – in the sense that all system components behave as intended – has to be abandoned. Open technical systems, as regarded in this thesis, are characterized by a large number of system components that have to work together to balance supply and demand in a very dynamic environment.¹ The latter results from the interactions between possibly volatile and heterogeneous system components. In some cases, the system components' behavior is driven by stochastic influences. This is the case with weather-dependent power plants or system components representing households, for example. These characteristics give rise to uncertainties that put the system at risk of not being able to balance supply and demand.

This thesis proposes an approach to *robust resource allocation in open technical systems*. We employ the methodology of *self-organization* to allow the system to form and maintain appropriate structures that tackle scalability issues and that support its objective under changing conditions. Furthermore, we make use of the principle of *computational trust* to measure and anticipate uncertainties at runtime. We

¹Another form of openness – often regarded in multi-agent systems research – is present when agents can arbitrarily enter and leave the system [56]. This form of openness is not part of the considerations of this thesis.

not only incorporate the gained information into the process of resource allocation itself, but also into the self-organized formation of adequate system structures to alleviate the influence of uncertainties. In the following, we highlight the necessity of this approach to cope with the challenges of future decentralized power systems, which serve as case study throughout this thesis. To begin with, we formalize the problem to be solved.

1.1 The Resource Allocation Problem

The problem we consider is an instance of the *single-resource allocation problem without externalities* [218].² The primary goal is to stipulate the supply $S_a[t]$ of *dispatchable* components $a \in \mathcal{V}$ in such a way that, in each point in time t , the sum $S_{\mathcal{V}}[t] = \sum_{a \in \mathcal{V}} S_a[t]$ of the dispatchable components' supply matches a demand $D[t]$ as accurately as possible. That is, the demand violation, which is expressed as the absolute difference $\Delta = |S_{\mathcal{V}}[t] - D[t]|$, should be minimal. The demand is imposed by a set of *non-dispatchable* components \mathcal{U} (the term “non-dispatchable” refers to components that cannot be controlled). While the satisfaction of the demand is paramount, this goal should be achieved at minimal costs Γ . The resource allocation problem can be specified in the form of a *constraint satisfaction optimization problem* (CSOP) [213] that is composed of a set of variables, their domains (note that we do *not* assume the domains to be finite, e.g., to allow for real-valued supply), constraints that restrict valid assignments, and an objective function as follows:

$$\underset{S_a[t]}{\text{minimize}} \quad \alpha_{\Delta} \cdot \Delta + \alpha_{\Gamma} \cdot \Gamma \tag{1.1a}$$

$$\begin{aligned} \text{subject to} \quad & \Delta = |S_{\mathcal{V}}[t] - D[t]|, \\ & \Gamma = \sum_{a \in \mathcal{V}} \kappa_a(S_a[t]), \end{aligned} \tag{1.1b}$$

$$\text{with} \quad S_{\mathcal{V}}[t] = \sum_{a \in \mathcal{V}} S_a[t]$$

In this optimization problem, the dispatchable components' supply $S_a[t]$ are the decision variables. The total costs Γ are based on component-specific cost functions κ_a that map a component's supply to costs (see Equation (1.1b)). We use the parameters α_{Δ} and α_{Γ} to establish the desired prioritization of the two objectives (see Equation (1.1a)).

In electric power systems, the system components comprise the set of dispatchable and non-dispatchable prosumers (we use the term “prosumer” to refer to producers as well as consumers). Further, the resource allocation problem's demand corresponds to the so-called *residual load*, which is defined as the difference between the overall non-dispatchable load that originates from, e.g., ordinary households, and the accumulated output of non-dispatchable power plants, such as solar power plants (see Figure 1.1). In other words, it is the fraction of the overall non-dispatchable load that has to be fulfilled by dispatchable prosumers. Note that we specify the output of non-dispatchable power plants to be part of the system's demand because their output cannot be controlled. Consequently, the resource allocation problem's supply is equivalent to the dispatchable prosumers' output. The fundamental goal is to find an allocation of the dispatchable prosumers' output such that, in each point in time t , their sum matches the residual load as accurately as possible. Further, the costs of satisfying the residual load should be kept low. Typical electricity production costs of biomass, hydro, and gas power plants are $17.50 \frac{\text{euro cent}}{\text{kWh}}$, $15.00 \frac{\text{euro cent}}{\text{kWh}}$, and $8.65 \frac{\text{euro cent}}{\text{kWh}}$, respectively [124].

When stipulating the supply $S_a[t]$ of a dispatchable component, we not only have to take account of maximal supply S_a^{\max} but also of a minimal (possibly positive) supply S_a^{\min} that results from technical or economical reasons (e.g., a minimum generation of 20 % of the nameplate capacity for gas turbines, or 40 % for coal based thermal plants [94]). We generalize the minimal and maximal boundaries S_a^{\min} , S_a^{\max} for the supply to *feasible regions* L_a represented by a finite list of non-overlapping intervals. This

²“Single-resource” refers to a particular type of resource, such as electric power. Further, “without externalities” means that a component's supply does not have detrimental side effects on another component's ability to contribute.

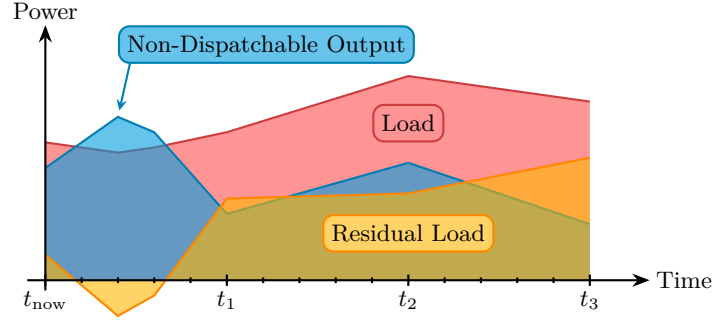


Figure 1.1: The residual load is the difference between the overall load of non-dispatchable consumers and the overall output of non-dispatchable power plants. It has to be satisfied by the dispatchable power plants and the dispatchable consumers. Note that the residual load is negative if the non-dispatchable output exceeds the non-dispatchable load.

allows us to take account of dispatchable components that can be turned off, such as gas turbines, causing a supply of $S_a[t] = 0$ that might be below the minimum of S_a^{\min} . A dispatchable component that can be turned off thus has feasible regions of $L_a = \langle [0, 0], [S_a^{\min}, S_a^{\max}] \rangle$. For instance, we write $L_a = \langle [0 \text{ MW}, 0 \text{ MW}], [4 \text{ MW}, 20 \text{ MW}] \rangle$ for a gas turbine that has to produce between 4 MW and 20 MW if turned on, and 0 MW if turned off. For components that cannot be turned off, such as so-called “must-run” power plants [230], we simply have $L_a = \langle [S_a^{\min}, S_a^{\max}] \rangle$.

Moreover, we have to assume that dispatchable components cannot arbitrarily change their supply within a fixed period of time. Instead, their behavior is subject to heterogeneous types of inertia – a property which can be found in many systems that control physical devices, such as power or heat generators. Inertia mainly manifests in a component-specific maximal rate of change [223]. As for thermal power plants, this is due to required heating and cooling. For example, typical gas, biomass, or coal power plants can change their output by $12 \frac{\% S_a^{\max}}{\text{min}}$, $6 \frac{\% S_a^{\max}}{\text{min}}$, or $3 \frac{\% S_a^{\max}}{\text{min}}$, respectively (in percent of the maximal supply S_a^{\max} per minute).³ Some power plants’ behavior is additionally subject to minimal cold and hot start-up or shut-down times that range – depending on the type of power plant – from a couple of minutes to several hours [21], as well as minimal or maximal uptimes [161].

Formally, we hereinafter abstract from continuous time by considering discrete *time steps* t . We define Δt as the distance between two successive time steps so that the time step $t + \Delta t$ is the successor of the time step t . We incorporate the dispatchable components’ control models into the resource allocation problem specified in Equation (1.1) by means of additional constraints (see Equations (1.2a) and (1.2b)). The property of inertia is represented by the functions \vec{S}_a^{\min} and \vec{S}_a^{\max} that restrict a dispatchable component’s supply $S_a[t]$ in time step t depending on its supply in the previous time step $t - \Delta t$.⁴ We

³<http://publica.fraunhofer.de/dokumente/N-364435.html>, retrieved on February 20, 2016.

⁴For convenience, we only model inertia that is subject to a component’s supply in the previous time step. To model more complex types of inertia, such as start-up times, further state variables, such as the component’s current uptime, have to be taken into account. We provide such a description in [187].

obtain the following thorough formulation of the resource allocation problem:

$$\begin{aligned}
& \underset{S_a[t]}{\text{minimize}} && \alpha_\Delta \cdot \Delta + \alpha_\Gamma \cdot \Gamma \\
& \text{subject to} && \forall a \in \mathcal{V} : \\
& && \exists [x, y] \in L_a : x \leq S_a[t] \leq y, && (1.2a) \\
& && \vec{S}_a^{\min}(S_a[t - \Delta t]) \leq S_a[t] \leq \vec{S}_a^{\max}(S_a[t - \Delta t]), && (1.2b) \\
& \text{with} && \Delta = |S_{\mathcal{V}}[t] - D[t]|, && (1.2c) \\
& && \Gamma = \sum_{a \in \mathcal{V}} \kappa_a(S_a[t]), \\
& \text{and} && S_{\mathcal{V}}[t] = \sum_{a \in \mathcal{V}} S_a[t]
\end{aligned}$$

Due to the components' inertia, it is not feasible to hold the balance between supply and demand by only *reactively* adapting the supply of dispatchable components. As the following example shows, their supply might not change quickly enough to reactively adapt to the demand in all situations: Let us consider two dispatchable power plants a_1 and a_2 , each with a maximal supply of $S_{a_1}^{\max} = S_{a_2}^{\max} = 20$ MW but different maximal rates of change of $2 \frac{\text{MW}}{\text{min}}$ and $5 \frac{\text{MW}}{\text{min}}$, respectively. With regard to the current time step t_{now} , we assume that the residual load corresponds to $D[t_{\text{now}}] = 36$ MW, a_1 supplies $S_{a_1}[t_{\text{now}}] = 10$ MW, and a_2 runs at its full capacity so that $S_{a_2}[t_{\text{now}}] = 20$ MW. Given this allocation of resources, we fall $\Delta = 6$ MW short in t_{now} . To reactively balance supply and demand, the power plants have to produce an extra 6 MW in the next time step $t_{\text{now}} + 1$ min, assuming that $D[t_{\text{now}} + 1 \text{ min}] = D[t_{\text{now}}]$. However, because a_1 and a_2 can only increase their output by 2 MW and 0 MW, respectively, we miss the target by 4 MW.

The dispatchable components' inertia is an important characteristic of the considered system class. In power systems, most power plants are therefore specialized to take on specific tasks: Peaking power plants, such as gas turbines, are able to adjust their output very quickly but cause high costs. On the other hand, base load power plants, such as coal power plants, are designed for operating very efficiently in specific output ranges, but their cold and hot start-up behavior as well as minimal and maximal up-times become additional decisive factors for control actions.

1.2 The Scheduling Problem – Proactive Resource Allocation to Address the Inert Provision of Resources

To take account of the different types of inertia, the supply of dispatchable components has to be specified *proactively* in the form of *schedules* for a fixed time span H in advance. This means that schedules are created on the basis of *predictions* of the future demand. A recalculation of the schedules is needed at least after the time span H elapsed since the last schedule creation. For each schedule creation, the time span H defines a discretized, so-called *scheduling window* $\mathcal{W} = \{t_{\text{now}} + i \cdot \Delta\tau \leq t_{\text{now}} + H \mid i \in \mathbb{N}_{\geq 1}\}$ that, depending on the *schedule resolution* $\Delta\tau$ (defined as a multiple of the difference Δt between two successive time steps t and $t + \Delta t$), comprises $N = H/\Delta\tau$ time steps.

Consequently, the system's success in solving the resource allocation problem (in terms of balancing supply and demand) depends on its success in solving a *scheduling problem*. In power systems, this problem is also known as *economic dispatch* [174] or *unit commitment* [161]. The idea of satisfying the demand by repeated schedule creation follows the principle of *model predictive control* and *receding horizon control* [128], which is illustrated in Figure 1.2.

Solving the scheduling problem requires solving the resource allocation problem specified in Equation (1.2) for all time steps in the scheduling window \mathcal{W} . We formalize the scheduling problem on the basis of a demand prediction $\hat{D}^{\mathcal{W}} = \langle \hat{D}^{\mathcal{W}}[t_{\text{now}} + \Delta\tau], \dots, \hat{D}^{\mathcal{W}}[t_{\text{now}} + H] \rangle$ and the set of dispatchable

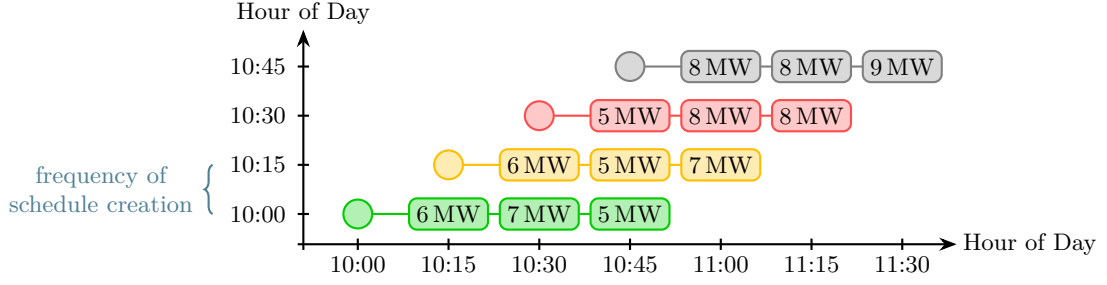


Figure 1.2: Schedules stipulate the dispatchable prosumers' output on the basis of residual load predictions (cf. the numbers in the boxes) for a specific time span in advance. Here, schedules cover a time span of $H = 45$ min and are created with a resolution of $\Delta\tau = 15$ min. Hence, each schedule comprises $N = 3$ time steps. At 10:00, for instance, schedules are created for the scheduling window $\mathcal{W} = \{10:15, 10:30, 10:45\}$. As predictions become more accurate as a future point in time approaches, schedules are periodically revised (here, every 15 min). Each schedule creation is depicted in a different color.

components \mathcal{V} as follows:

$$\underset{S_a[t]}{\text{minimize}} \quad \alpha_\Delta \cdot \Delta + \alpha_\Gamma \cdot \Gamma \quad (1.3a)$$

$$\text{subject to} \quad \forall a \in \mathcal{V}, \forall t \in \mathcal{W} : \quad (1.3b)$$

$$\begin{aligned} \exists [x, y] \in L_a : x \leq S_a[t] \leq y, \\ \vec{S}_a^{\min}(S_a[t - \Delta\tau]) \leq S_a[t] \leq \vec{S}_a^{\max}(S_a[t - \Delta\tau]), \end{aligned}$$

$$\text{with} \quad \Delta = \sum_{t \in \mathcal{W}} |S_{\mathcal{V}}[t] - \hat{D}^{\mathcal{W}}[t]|, \quad (1.3c)$$

$$\Gamma = \sum_{a \in \mathcal{V}, t \in \mathcal{W}} \kappa_a(S_a[t]), \quad (1.3d)$$

$$\text{and} \quad S_{\mathcal{V}}[t] = \sum_{a \in \mathcal{V}} S_a[t]$$

The differences between the scheduling problem and the resource allocation problem specified in Equation (1.2) can all be ascribed to the regarded scheduling window \mathcal{W} : First, the supply is stipulated in the form of schedules $S_a = \langle S_a[t_{\text{now}} + \Delta\tau], \dots, S_a[t_{\text{now}} + H] \rangle$ for each $t \in \mathcal{W}$ (see Equation (1.3b)). Second, the total demand violation Δ is defined as the sum of the absolute deviations between the total scheduled supply $S_{\mathcal{V}}[t]$ and the predicted demand $\hat{D}^{\mathcal{W}}[t]$, calculated over all $t \in \mathcal{W}$ (see Equation (1.3c)). By summing up *absolute* violations, negative and positive deviations cannot cancel each other out. Third and analogously to the total demand violation, the total costs Γ are also calculated over all $t \in \mathcal{W}$ (see Equation (1.3d)).

1.3 Challenges of Resource Allocation in Open Technical Systems

Solving the scheduling problem introduces two central and interconnected challenges:

1. **Scalability:** The scheduling problem is a computation-intensive combinatorial optimization problem (typically formulated as a mixed integer linear program) that is NP-hard with regard to the number $|\mathcal{V}|$ of dispatchable components involved and time steps N schedules are created for in advance [29].⁵ We therefore have to assume a worst-case complexity of $O(2^{|\mathcal{V}| \cdot N})$, which is why exact scheduling algorithms do not scale well.

⁵Since dispatchable components show discrete modes of operation (e.g., on/off), the knapsack problem (i.e., choosing which components should contribute at all) can be seen as a special case of the scheduling problem.

2. **Uncertainty:** The dispatchable components have to fulfill the demand in spite of erroneous predictions of its development and unexpected events. In power systems, imprecise load predictions mainly originate from inappropriate or outdated standard load profiles that do not match the behavior of specific consumers, e.g., households possessing electric vehicles or solar panels [81, 86].⁶ The quality of the weather-dependent power plants' predicted output is impaired by imprecise sensor data and inaccurate weather forecasts for specific geographic locations. Stochastic influences, such as changing weather conditions and stochastic consumer behavior, further lead to fluctuations in the residual load. As for unexpected events, dispatchable prosumers might not be able to comply with their schedules in the light of technical difficulties. In terms of the resource allocation problem, uncertainties manifest in the form of deviations between actual and predicted demand, as well as deviations between actual and scheduled supply.

With regard to scalability, a too fine-grained schedule resolution $\Delta\tau$, which unnecessarily increases the number of time steps N in the scheduling window \mathcal{W} , should be avoided due to the exponent N in the scheduling problem's computational complexity. In the synchronous grid of Continental Europe, for instance, power plant schedules are typically created with a resolution of $\Delta\tau = 15$ min (cf. Figure 1.2), whereas imbalances have to be detected and compensated for within seconds (e.g., $\Delta t = 5$ s) to ensure the grid's stable operation [214].⁷ At the same time, a too fine-grained schedule resolution is not useful because, given that demand predictions tend to become more accurate as a future point in time approaches, uncertainties require that schedules are periodically revised at runtime.

For many years, power systems consisted of relatively few and well-predictable power suppliers – coal and nuclear power plants for the most part – that faced a large number of pure power consumers. Especially in Germany, the deregulation of the electricity market, climate protection goals, as well as the enacted nuclear power phase-out⁸ have been just a few of various driving forces changing this situation. Nowadays, the wide-spread installation of weather-dependent power plants as well as the advent of new consumer types, such as electric vehicles, put a lot of strain on power grids [173]. Additionally, small potentially dispatchable power plants (e.g., biogas plants) owned by individuals or cooperatives feed in power without external control. To deal with this situation, the German transmission system operators Tennet and 50 Hertz spent about 1 billion € to stabilize the power grid in 2015. Compared to 2014, Tennet's total expenses increased by 240 %.⁹ The electric utility EWE reported that the network-stabilizing measures increased by 700 % from 2009 to 2011 in order to comply with the supply of wind turbines and biogas power plants.¹⁰ Regarding the number of power plants subsidized by the German Renewable Energy Act (EEG)¹¹, we observe an increase of 113 % from 734545 in 2010 to 1565154 in 2015. The amount of installed weather-dependent output raised by 112 %. In the same time frame, the number of *potentially* dispatchable power plants (including the biogas power plants mentioned above) grew by 39 % and their annual feed-in by 51 %, though. Table 1.1 provides a detailed overview of these numbers (source EnergyMap.info¹²). These trends will endure, taking the current plans into account (cf. the Europe 2020 strategy¹³).

⁶Standard load profiles define a presumed consumption of a specific type of consumer (e.g., household or small business) for a specific time of day depending on the day of the week and the season. They usually exist in a resolution of 15 min. They are based on aggregated statistical data.

⁷In the formalization of the optimization problems regarded in this thesis, we assume that algorithms solving the problem are fast enough to provide a solution within the specified period of time, e.g., 5 s when compensating for imbalances of 500 MW in the power grid. If such an algorithm is not able to provide a feasible solution in time, it cannot be applied to the regarded problem, or the specified period of time has to be increased.

⁸See "Act on the Peaceful Utilisation of Atomic Energy and the Protection against its Hazards (Atomic Energy Act)" by the Bundesamt für Strahlenschutz, available online at http://www.bfs.de/SharedDocs/Downloads/Bfs/EN/hns/a1-english/A1-01-16-AtG.pdf?__blob=publicationFile&v=7. Retrieved on February 20, 2016.

⁹<http://www.heise.de/newsticker/meldung/Kampf-gegen-Stromnetz-Blackout-Rekordkosten-von-einer-Milliarde-Euro-3072872.html>, retrieved on February 20, 2016.

¹⁰http://www.nwzonline.de/hintergrund/ewe-spuert-wende-deutlich_a-1,0,519127305.html, retrieved on February 20, 2016.

¹¹http://www.gesetze-im-internet.de/eeg_2014/index.html, retrieved on February 20, 2016.

¹²<http://www.energymap.info>, retrieved on February 20, 2016.

¹³<http://www.bmwi.de/English/Redaktion/Pdf/eu-kommission-mitteilung-europa-2020,property=pdf,bereich=bmwi2012,sprache=en,rwb=true.pdf>, retrieved on February 20, 2016.

	2010			2015		
	Number	Peak [MW]	GWh/year	Number	Peak [MW]	GWh/year
Solar	697291	12297	11748	1515063	37465	34672
Wind	20130	26053	45290	26206	43965	77012
Total Non-Dispatchable	717421	38350	57038	1541269	81430	111684
Water	7030	1441	5644	7513	1656	6732
Biomass	9246	4272	24308	15499	7156	40003
Geothermal	5	8	40	29	33	125
Others	843	683	2481	844	635	2257
Total Dispatchable	17124	6404	32472	23885	9480	49117
Total	734545	44754	89511	1565154	90910	160801
Fraction Dispatchable	2.33 %	14.31 %	36.28 %	1.53 %	10.43 %	30.55 %

Table 1.1: The number, peak output, and annual feed-in of power plants subsidized by the German Renewable Energy Act (EEG) in 2010 and 2015 according to EnergyMap.info: From 2010 to 2015, these numbers increased by 113 %, 103 %, and 80 %, respectively. As for non-dispatchable power plants, we even observe an increase of 115 %, 112 %, and 96 %, respectively. But also the share of dispatchable power plants grew significantly by 39 %, 48 %, and 51 %, respectively. In the same time frame, the consumption of electricity maintained more or less constant (604950 GWh/year in 2010, compared to 608051 GWh/year in 2015). Hence, the fraction of the consumption satisfied by EEG-subsidized power plants raised from 15 % to 26 %, which is an increase of 79 %.

To save expenses, gain more flexibility, and deal with uncertainties, future *autonomous* power systems have to take advantage of the potentially dispatchable prosumers by incorporating them into the scheduling scheme. Besides scalable approaches to schedule creation, the ability to deal with uncertainties introduced by non-dispatchable prosumers becomes a major concern. Although the output of weather-dependent power plants is difficult to predict, simply turning them off is not feasible because the system might depend on their resources at on-peak hours and benefits from their low-cost generation. Utilizing the output of renewable energy sources is further incentivized by legal regulations, such as the EEG. Therefore, to ensure the system’s stable and efficient operation, uncertainties have to be anticipated when creating schedules and have to be compensated for locally to prevent their propagation through the system.

In this thesis, we propose to deal with these challenges by the principles of *self-organization* and *computational trust*.

1.4 Self-Organization, Computational Trust, and Robustness

In this section, we briefly introduce the principles of self-organization and computational trust. On this basis, we outline the two dimensions of robustness considered in this thesis. A detailed discussion of the properties of computational trust can be found in Part II. Part III presents different approaches to self-organization in the context of this thesis.

Self-Organization The capacity to self-organize will be of the utmost importance to master the challenges of future power systems, in particular, their inherent resource allocation problem of balancing supply and demand. Self-organizing systems are characterized by their ability to autonomously adapt their (organizational) structure in response to external disturbances or to changes in their internal state; thus maintaining the system’s functionality (cf. [23, 195, 196]). An important aspect is that self-organizing systems are robust to failures given that their flexibility allows them to cope with a wide range of circumstances in uncertain environments [88]. Instead of having human operators constantly dealing with ongoing issues, the systems’ ability to autonomously maintain their functionality is implemented in self-* algorithms. An organizational structure that ensures the system’s functionality and supports its objectives not only promotes productivity and robustness, but also ensures the scalability of the

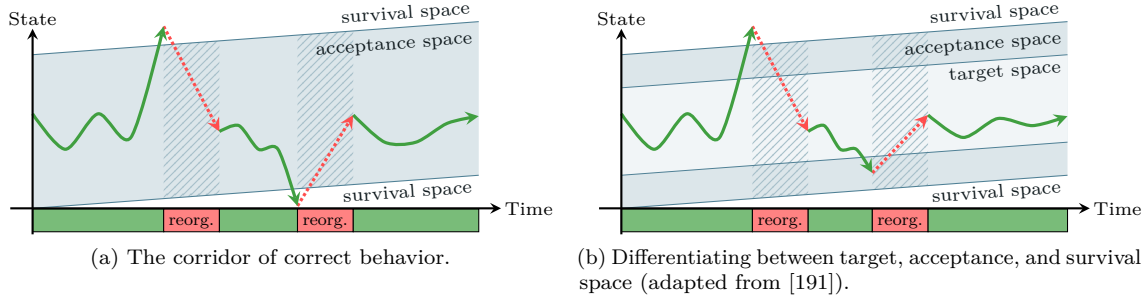


Figure 1.3: Once a violation of the corridor of correct behavior is detected, the system triggers a reorganization that re-establishes compliance with the invariant (see Figure 1.3a). Refining the corridor by means of a target space allows the system to preserve its efficiency by triggering a reorganization before it leaves the acceptance space (cf. the second reorganization in Figure 1.3b). In this case, reorganizations aim at bringing the system back into an optimal state.

system [92]. The principle of self-organization has been successfully adopted to a variety of applications, such as flexible manufacturing systems, robotics, and many-core systems (see [151] for an overview). Self-organizing systems are often implemented on the basis of a *multi-agent system* (MAS) [228].

To allow for self-organization, the system has to be provided with degrees of freedom. Due to the given flexibility, a major challenge is to control the system's behavior [191]. The *Corridor Enforcing Infrastructure* (CEI) [63] is an architectural pattern to ensure that the system fulfills its requirements – that have been identified at design time – at runtime. Within the CEI, the concepts of the *Restore Invariant Approach* [153] are applied. In the Restore Invariant Approach, an invariant (a logical formula) separates invalid from valid states, which results in a so-called *corridor of correct behavior* (see Figure 1.3a). Structurally, the invariant is a conjunction of *constraints* that have to be satisfied by the system variable assignment induced by a state. If the invariant is satisfied, the system is inside the corridor. The system is outside the corridor if the invariant is violated (i.e., at least one constraint is not satisfied). As soon as such a violation is detected, a reorganization is initiated that guides the system back into the corridor. A governing idea is that the system delivers its *correct* functionality as long as the invariant over the individual components and the organizational structure holds. Similarly to the corridor of correct behavior, Schmeck et al. [191] proposed a distinction of states depicted in Figure 1.3b; they define the *acceptance space* (i.e., correct states), the *target space* (i.e., optimal states), and the *survival space* (i.e., states outside the corridor that allow for a reorganization leading the system back into the corridor). The *dead space* defines the set of irreversibly faulty states.

The CEI includes decentralized feedback loops to monitor the system's state and to trigger a reorganization when necessary. These feedback loops implement the Observer/Controller (O/C) architecture [175], which is similar to the MAPE cycle [110]. While the observer monitors the state of the *System under Observation and Control* (SuOC), the controller uses information provided by the observer to decide about necessary control actions. The controller uses self-organization algorithms to reorganize the SuOC. It also makes sure that the result of the self-organization algorithms is adopted by the SuOC.

Based on the distinction between the acceptance space and the target space, Schmeck et al. [191] define a system to be *strongly robust* if the system remains in the target space despite disturbances. In this thesis, we focus on *weakly robust* systems, i.e., systems that, once being in the target space, cannot be forced to leave the acceptance space by external disturbances. Here, the acceptance space refers to states in which supply and demand are adequately balanced. Clearly, weak robustness cannot be achieved by putting the system in an arbitrary valid state that might itself verge on the edge of acceptance. This would allow even small disturbances to move the system out of the acceptance space. Instead, the system has to solve multiple interconnected optimization problems (e.g., finding optimal system structures or optimally scheduling control actions) in order to find states placed well inside the target space that allow the system to deal with a wide range of disturbances.

Computational Trust In open self-organizing systems, the agents operate in a highly uncertain environment in which their interaction partners' benevolence cannot be assumed. In multi-agent systems research, *computational trust* (cf. [150]) has been presented as a means to cope with uncertainties stemming from unintentional as well as intentional misbehavior. The concept of trust is strongly connected to *risk* [122]. Basically, trust mirrors expectations about others [55]. It is based on experiences [100] that stem from contracts in which two or more parties stipulate a desired result of an interaction [172]. Since trust represents an empirically justified expectation, it reduces uncertainties and risks associated with another agent's behavior [171]. By preferring interactions with trustworthy agents, trust-based decisions incentivize rational agents to comply with their contracts [26]. That way, the system's efficiency can be increased. In computing systems, trust is often captured by a numerical *trust value* [141]. In some trust models (see, e.g., [112, 226]), additional measures are used to indicate the belief that the trust value accurately describes an agent's actual observable behavior.

Robustness Due to these discussed characteristics, self-organization and computational trust are necessary principles to enable robust resource allocation in large-scale open technical systems. In this thesis, we consider the following two dimensions of robustness: The first dimension of robustness addresses a system's ability to resist internal or external disturbances (cf. [46]). For instance, such disturbances might result from (un)intentionally misbehaving agents, e.g., those whose wrongdoing is driven by changing environmental conditions. A system exhibiting this type of robustness promises to remain in acceptable states and thus to maintain its functionality despite detrimental influences. The second dimension of robustness considers a system's ability to return into an acceptable state after a disturbance occurred that caused the system to leave the acceptance space. This type of robustness characterizes a system's ability to restore its functionality. Both dimensions of robustness quantify the system's ability to fulfill its tasks. In contrast to a mere passive resistance, self-organizing systems can actively increase their robustness by means of reactive or proactive measures. In open systems, these measures can be based on the participants' trustworthiness, which allows the system to anticipate different sources of uncertainties.

In the next chapter, we introduce our vision of robust and scalable autonomous power systems that is based on the principles of self-organization and computational trust.

Decentralized Power Management

Summary. In this thesis, we illustrate our findings by means of a case study from the field of decentralized autonomous power systems. As discussed in Chapter 1, future power systems will be exposed to a high degree of uncertainty introduced by weather-dependent power plants and new types of consumers. Further, they will be characterized by a vast number of potentially dispatchable power plants. To maintain the balance between power production and consumption, uncertainties have to be anticipated and advantage has to be taken of the dispatchable power plants. Based on the principles of self-organization and computational trust, we present our concept of *Autonomous Virtual Power Plants* to master these challenges.

Publication. The case study as well as the overview of current and future power systems have been published in Anders et al. [6] and Steghöfer et al. [205].

Decentralized power systems are a prominent example of the class of open technical systems introduced in Chapter 1. Section 2.1 explains the current control of power systems and gives an overview of different approaches to autonomous power management. We introduce our concept of *Autonomous Virtual Power Plants* (AVPPs) to meet the challenges of future power systems in Section 2.2. This concept embodies our vision of self-organized and robust resource allocation in large-scale open technical systems. Throughout this thesis, the challenge of balancing power production and consumption in decentralized autonomous power systems illustrates our findings and serves to evaluate the devised concepts and algorithms. An overview of related projects in the domain of smart grids is given in Section 2.3.

2.1 Current and Future Power Systems

In current power systems, electric utilities only create schedules for large dispatchable power plants, such as waste-to-energy, coal, and nuclear power plants. Schedules are created in a centralized manner, e.g., by solving the scheduling problem defined in Section 1.2 by means of the standard mathematical programming software IBM ILOG CPLEX¹. As stated in Section 1.3, a very large number of small *potentially* dispatchable power plants (e.g., micro combined heat and power units or biogas plants) owned by individuals or cooperatives generate and feed in power without the electric utilities' control. Because very few measuring equipment is available in the field today, the actual status of the grid, generators, and consumers is usually unknown. Furthermore, predictions used for schedule creation are based on standard load profiles, weather forecasts, historical analysis, and intuition instead of current, reliable data [18]. Nowadays, they are made by humans or SCADA (Supervisory Control and Data Acquisition) systems.

¹IBM ILOG CPLEX Optimizer, Version 12.4, 2011: <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, retrieved on March 1, 2016.

To compensate for this lack of control, additional power is provided in the form of the operating reserve. In the synchronous grid of Continental Europe, operators hold back an amount of 3 GW of idle power that can be activated within seconds to maintain the balance between power production and consumption [214].² To account for the compensation for imbalances, the power grid is structured into balancing groups. Electric utilities usually take on the role of balancing group managers, i.e., the institutions in charge.

However, the de facto standard of centralized control becomes more and more impracticable, e.g., due to the enormous increase in costs. This is clearly confirmed by the numbers we discussed in Section 1.3. To preserve the stability and ensure economic efficiency of the future power system, we have to take advantage of the potentially dispatchable power plants by integrating them into the scheduling scheme. Further, schedules have to be created on the basis of the up-to-date predictions of the future residual load and information about the current status of the grid. Many studies and researches agree (e.g., [18, 143, 173, 201]) that a scalable and stable operation of future grids hinges on the idea of enabling the system components to act autonomously and proactively in their environment. The integration of modern information and communications technology (ICT) into the power grid – such as broadband communications as well as automated monitoring and control infrastructure (e.g., smart meters) – has been identified as key enabler to allow for such a transition to *smart grids* [18, 158].

Virtual power plants (VPPs) are a fundamental concept to deal with the large scale of future power systems [18, 173]. VPPs are groups of power plants or consumers that are usually controlled by a central entity (see, e.g., [136]). Sometimes, membership in a VPP is restricted to certain types of power plants with predefined properties, such as dispersed generation units and micro combined heat and power units in [192] or distributed energy resources in [31]. These approaches mainly focus on providing structures to integrate distributed energy resources or consumers into existing control schemes. For the most part, VPPs are used to aggregate the capacity of multiple producers or consumers to overcome market barriers and to facilitate trading in power markets [157, 176]. On the intraday market of the European Power Exchange SE³ in Germany, for instance, contracts are concluded with a minimum volume increment of 100 kW for individual time spans of 15 min⁴ in which the stipulated volume has to be constantly delivered or accepted. VPPs allow for a constant output or load, which cannot be achieved by an individual consumer, distributed energy resource, or weather-dependent power plant. In such a context, the approach presented in [168] distinguishes commercial VPPs that participate in power markets from technical VPPs that provide services for the transmission network.

While some of these approaches allow for a self-organized formation of VPPs (e.g., [157, 176]), none of them combines all the features that we are looking for to maintain the balance between power production and consumption in future power systems. This can be ascribed to the fact that the *formation* of VPPs for power markets does not aim for system structures that allow for the scalable creation of high-quality schedules and that improve the robustness and the efficiency of the *overall* system. Still, these approaches give important insights into the organization and functionality of such a system. In the following section, we present our vision of self-organizing autonomous power systems.

2.2 Autonomous Virtual Power Plants – a Trust-Based Multi-Agent Approach to Self-Organizing Autonomous Power Management

In contrast to the above-mentioned approaches, we consider the problem of autonomously maintaining the balance between power production and consumption. To accomplish this task, the full potential of dispatchable power plants has to be exploited by incorporating them into the scheduling scheme. This calls for a scalable solution. Further, uncertainties have to be anticipated when creating schedules and compensated for locally to prevent their propagation through the system.

²In the power grid, imbalances manifest in deviations from the nominal frequency 50 Hz. If too much power is fed into the system, the frequency is above 50 Hz. In case of a deficit, the frequency is below 50 Hz. The relation between the mismatch between power production and consumption and the utility frequency is given by the network frequency characteristic (15000 MW/Hz for Continental Europe) [214]. The frequency must be kept between 49.8 Hz and 50.2 Hz.

³<https://www.epeaxspot.com/en/>, retrieved on February 20, 2016.

⁴<https://www.epeaxspot.com/en/product-info/intradayauction/germany>, retrieved on February 20, 2016.

We adopt the natural approach to design autonomous power systems by representing each prosumer by a software agent that is able to communicate with the other agents in the system. That way, we enable prosumers to proactively participate in the creation of schedules and in maintaining the stability of the grid.

Our approach to meet the challenges of future power systems is called *Autonomous Virtual Power Plants* (AVPPs). AVPPs tackle the challenges of scalability and uncertainty based on the principles of self-organization and computational trust. AVPPs represent self-organizing groups of two or more power plants of various types comprising both dispatchable as well as non-dispatchable power plants (e.g., biogas, hydro, and solar power plants). Each AVPP has to satisfy a fraction of the overall residual load by periodically calculating schedules for its dispatchable power plants on the basis of residual load predictions. The overall residual load stems from the AVPPs' *local demand*, i.e., its local residual load. Note that an AVPP's local residual load originates from its non-dispatchable consumers as well as its non-dispatchable power plants. Depending on the AVPP's composition, the local residual load is either positive or negative. The local residual load is negative, for instance, if the AVPP does not contain any consumers but non-dispatchable power plants, which leads to a surplus of production. The local residual load is positive if the consumption of its consumers exceeds the output of its non-dispatchable power plants. To balance production and consumption, an AVPP's dispatchable power plants have to reactively compensate for deviations between the actual and the predicted local residual load as well as deviations between actual and scheduled output. By compensating for deviations locally, AVPPs avoid that deviations propagate to and affect other parts of the system. To reduce the influence of erroneous residual load predictions and schedule violations, AVPPs use trust models to measure and anticipate the accuracy of residual load predictions as well as the compliance with created schedules. A prosumer's trustworthiness is the higher, the lower and the less varying its deviations are.

To cope with the vast number of dispatchable power plants, AVPPs self-organize into a *hierarchical* structure.⁵ That way, we decompose the overall set of dispatchable \mathcal{V} and non-dispatchable \mathcal{U} power plants $\mathcal{A} = \mathcal{V} \cup \mathcal{U}$ into several hierarchically arranged AVPPs (see Figure 2.1). In this hierarchy, each AVPP acts as an *intermediary* $\lambda \in \mathcal{I}$ between its superordinate AVPP and its subordinate⁶ power plants $\mathcal{A}_\lambda \subset \mathcal{A}$. Since an intermediary λ represents the subsystem \mathcal{A}_λ , it can be viewed and treated as one large (dispatchable) power plant that subsumes the behavior of its collective (thus $\mathcal{I} \subset \mathcal{V} \subseteq \mathcal{A}$). As the hierarchy constitutes a tree, the sets of subordinate power plants are pairwise disjoint. We refer to the root of the hierarchy as the *top-level intermediary/AVPP* $\Lambda \in \mathcal{I}$. While we focus on power generation in this thesis, both dispatchable and non-dispatchable power consumers could be integrated into our concept of AVPPs and in the presented techniques. In the following, we assume that all consumers are assigned to the top-level intermediary, that they do not change their affiliation, and that schedules are only created for dispatchable power plants.

In this hierarchical system structure, AVPPs autonomously create schedules in a regionalized and top-down manner, meaning that each AVPP λ redistributes its own assigned residual load, i.e., its scheduled supply S_λ , to its subordinate dispatchable power plants \mathcal{V}_λ (note that these might contain subordinate AVPPs because $\mathcal{I}_\lambda \subset \mathcal{V}_\lambda$). The top-down creation of schedules is triggered by the top-level intermediary Λ whose assigned residual load S_Λ corresponds to its local residual load prediction $\hat{D}_\Lambda^{\mathcal{W}}$ (i.e., $S_\Lambda = \hat{D}_\Lambda^{\mathcal{W}}$). In the hierarchy, an AVPP's local residual load originates from its non-dispatchable prosumers and *additionally* contains the local residual load of its subordinate intermediaries. The local residual load prediction of the top-level AVPP is therefore equivalent to the overall predicted residual load. We lift the centralized scheduling problem defined in Equation (1.3) to hierarchical structures in

⁵Note that our approach currently does not take account of constraints imposed by the transmission network, such as line capacities or voltage bounds [35]. We leave this challenge for future work.

⁶“Subordinate” power plants are those an AVPP is *directly* responsible for, i.e., those on its next lower level in the hierarchy.

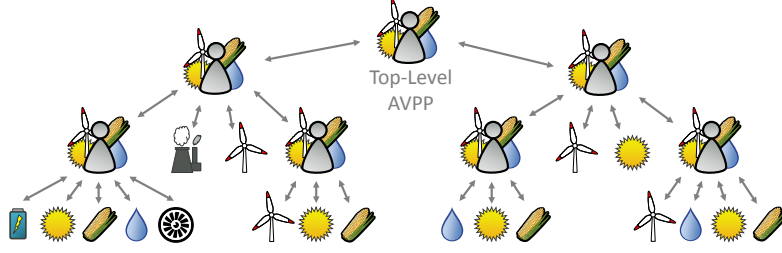


Figure 2.1: Hierarchical system structure of a future autonomous and decentralized power system: Power plants are structured into systems of systems represented by AVPPs that act as intermediaries to decrease the complexity of control and scheduling. AVPPs can be part of other AVPPs. The left child of the top-level AVPP, for instance, controls a coal power plant, a wind turbine, and two subordinate AVPPs. The leftmost AVPP, on the other hand, controls only physical power plants; from left to right: a storage battery as well as a solar, a biomass, a hydro, and a gas turbine power plant.

which intermediaries create schedules in a top-down manner as follows:

$$\underset{S_a[t]}{\text{minimize}} \quad \alpha_\Delta \cdot \Delta + \alpha_\Gamma \cdot \Gamma \quad (2.1a)$$

$$\text{subject to} \quad \forall a \in \mathcal{V}_\lambda, \forall t \in \mathcal{W} : \quad (2.1b)$$

$$\exists [x, y] \in L_a^t : x \leq S_a[t] \leq y, \quad (2.1b)$$

$$\vec{S}_a^{\min}(S_a[t - \Delta\tau]) \leq S_a[t] \leq \vec{S}_a^{\max}(S_a[t - \Delta\tau]),$$

$$\text{with} \quad \Delta = \sum_{t \in \mathcal{W}} |S_{\mathcal{V}_\lambda}[t] - S_\lambda[t]|, \quad (2.1c)$$

$$\Gamma = \sum_{a \in \mathcal{V}_\lambda, t \in \mathcal{W}} \kappa_a(S_a[t]), \quad (2.1d)$$

$$\text{and} \quad S_{\mathcal{V}_\lambda}[t] = \sum_{a \in \mathcal{V}_\lambda} S_a[t] \quad (2.1e)$$

The hierarchical scheduling problem differs from the centralized scheduling problem in three points: First, the demand that has to be distributed corresponds to an intermediary's scheduled supply S_λ (see Equation (2.1c)). Second, an intermediary creates schedules for its subordinate dispatchable agents \mathcal{V}_λ instead of for all dispatchable agents \mathcal{V} at once (see Equations (2.1a) and (2.1c) to (2.1e)). Third, we have to assume that a subordinate dispatchable agent's feasible supply ranges L_a^t depend on the considered time step $t \in \mathcal{W}$ (see Equation (2.1b)). That is because an intermediary λ uses an abstracted control model for each subordinate intermediary $\lambda' \in \mathcal{I}_\lambda$ when creating schedules in order to profit from hierarchical problem decomposition (more details concerning this matter can be found at the beginning of Chapter 6).

Regarding Equation (2.1), each AVPP can create schedules for its subordinates by means of a centralized solver. For example, instances of the hierarchical scheduling problem can be formulated as mixed integer linear programs that can be solved by IBM ILOG CPLEX. In the following, we refer to the approach in which each AVPP uses a centralized solver to *assign* schedules to subordinates as the *regio-central approach* (*RegioC*). Even in this situation, the complexity of solving the scheduling problem is reduced because each AVPP controls only a subset \mathcal{V}_λ of all dispatchable power plants \mathcal{V} . While the complexity of the centralized scheduling problem is $O(2^{|\mathcal{V}| \cdot N})$ (see Section 1.3), we now have a complexity of $O(2^{|\mathcal{V}_\lambda| \cdot N})$ for each AVPP $\lambda \in \mathcal{I}$. Compared to $|\mathcal{V}|$, AVPPs thus have to deal with smaller exponents $|\mathcal{V}_\lambda|$. Assuming perfect parallelization, the time needed to solve the overall scheduling problem corresponds to the longest of all serial paths that result from adding the AVPPs' scheduling times for each branch in the hierarchy (a formal definition is provided in Equation (6.1)). Given an adequate hierarchical structure, this results in shorter scheduling times for the overall system in comparison to a centralized approach.

While hierarchical problem decomposition has been proposed as a generic approach to deal with large-scale systems prohibiting a centralized solution [2, 40], our self-organizing hierarchy of AVPPs is a means to *autonomously* decompose the scheduling problem *at runtime*. Our main concern is to obtain a scalable structure of AVPPs that allows each of them to create high-quality schedules, and to locally deal with uncertainties stemming from non-dispatchable power plants as well as non-dispatchable consumers. Because each AVPP pursues the same goal – that is, to satisfy the assigned residual load as accurately and cost-efficiently as possible –, AVPPs aim at forming organizations that are, with respect to certain criteria, as similar as possible. An example of that are AVPPs that feature a similar ratio between typical variations in prediction errors of their non-dispatchable prosumers (this is reflected in their trust values) and degrees of freedom in terms of the controllability of their dispatchable power plants. That way, each AVPP is equally able to compensate for local deviations between power production and consumption, which improves the robustness of the overall system. Note that due to the system’s heterogeneity, these similar AVPPs are likely to be composed of dissimilar power plants. An overview of the different formation criteria of AVPPs is given in Section 6.3. AVPPs autonomously reorganize their structure to maintain an appropriate compromise between the quality of schedules and the time needed for their creation, and to be able to hold the balance between power production and consumption despite changing AVPP-internal or environmental conditions. A reorganization of a specific region within the hierarchy is triggered if it does not sufficiently satisfy one or more formation criteria. In particular, an AVPP triggers a reorganization if it cannot repeatedly compensate for its local uncertainties.

Such a setting imposes several challenges that we address in this thesis: (1) To quantify and anticipate uncertainties stemming from different sources, AVPPs have to create predictive models of their subordinate prosumers. In our approach, we achieve this by means of elaborate trust models. (2) To allow for the formation of system structures that feature an appropriate trade-off between scheduling times, the schedules’ quality, and the system’s stability, new types of organizations have to be considered when forming AVPPs. (3) To create and maintain such structures at runtime, new self-organization algorithms have to be developed. (4) To promote the system’s robustness, the scheduling problem has to be devised in a way that allows for anticipating uncertainties and reactively compensating for deviations at runtime. To find accurate solutions to these problems in large systems in appropriate time, new scheduling algorithms have to be employed.

2.3 Smart Grid Projects

The contributions of this dissertation were developed in the context of the research unit OC-Trust (FOR 1085)⁷ of the German Research Foundation, which set out in 2009. Since then, different challenges of future smart grids have been and are investigated in several projects, which we briefly present in the following. Detailed discussions of their research results related to the contributions of this thesis can be found in the corresponding sections. Many of these projects concentrate on demand-side management, demand response, the integration of distributed energy resources into power markets, and the proper operation of the power grid infrastructure, including the provision of ancillary services, such as reactive power and voltage control. In the context of the *E-Energy*⁸ program, multiple projects, such as *MeRegio*⁹ and *eTelligence*¹⁰, focused on the development of new market places. Based on the market place devised in the MeRegio project, the project *MeRegioMobil*¹¹ investigated the integration of electric vehicles into the grid. The project *Smart Watts*¹² considered the integration of power consumers into power markets in the sense of demand response. In the *Future Energy Systems*¹³ project, techniques for transmission and power system restoration after significant disruptions, and new centralized optimization platforms

⁷<http://www.isse.uni-augsburg.de/projects/reif/oc-trust/>, retrieved on February 20, 2016.

⁸<http://www.e-energy.de>, retrieved on February 20, 2016.

⁹<http://www.meregio.de>, retrieved on February 20, 2016.

¹⁰<http://www.etelligence.de>, retrieved on February 20, 2016.

¹¹<https://mergiomobil.forschung.kit.edu>, retrieved on February 20, 2016.

¹²<http://www.smartwatts.de>, retrieved on February 20, 2016.

¹³<http://org.nicta.com.au/portfolio/energy-systems/>, retrieved on February 20, 2016.

are developed. The project *quanticol*¹⁴ develops smart distribution grids with a focus on power flows, storages, and voltage control. The *FENIX*¹⁵ project examined the inclusion of small distributed energy resources into the grid by aggregating them into a *virtual power plant* (VPP) that acts on the market as a single entity. In *ADDRESS*¹⁶, the architecture developed in FENIX is used to enable active demand, i.e., the active participation of small consumers, such as households. Similarly, *Smart Nord*¹⁷ investigates the decentralized provision of reserves and active power products in distribution grids with a focus on consumers and distributed energy resources. In FENIX, ADDRESS, and Smart Nord, VPPs are used to overcome market barriers (see Section 2.2).

All these projects tackle the challenges of future smart grids from individual perspectives and on different levels of abstraction, thereby investigating specific aspects. We are convinced that a combination of the results of these projects – together with the contributions of this thesis – are important and necessary steps towards scalable, robust, and flexible power systems of the future.

¹⁴<http://blog.inf.ed.ac.uk/quanticol/smart-grids/>, retrieved on February 20, 2016.

¹⁵<http://www.fenix-project.org>, retrieved on February 20, 2016.

¹⁶<http://www.addressfp7.org>, retrieved on February 20, 2016.

¹⁷<http://smarnord.de>, retrieved on February 20, 2016.

Contributions of this Thesis: Self-Organization and Computational Trust for Scalable and Robust Resource Allocation under Uncertainty

Summary. Gives a brief overview of this thesis and lists its contributions. The concepts, approaches, and algorithms that form the contributions of this thesis have been published by the author in various peer-reviewed conferences, journals, and workshops. A complete list of publications of the author can be found in the back matter on page 261.

The goal of this thesis is to provide an integrated approach to *robust resource allocation in open technical systems*. To meet the challenges imposed by the considered system class, it combines the principles of *self-organization* and *computational trust* as well as techniques from the field of *online stochastic optimization*. To quantify and anticipate uncertainties, we devise predictive models of the agents' possibly deviating behavior (see Part II). To cope with the large scale of the regarded systems, we propose an organizational structure that supports the system's stability, scalability, efficiency, and robustness (see Part III). To deal with uncertainties when solving the resource allocation problem, we specify characteristics of robust solutions to the scheduling problem and introduce a mechanism that allows the agents to create such solutions in adequate time (see Part IV). The thesis comprises the following contributions:

A Trust Model to Quantify and Anticipate Uncertainties We present the concept of *Trust-Based Scenario Trees* (TBSTs) as an approach to probabilistic model creation. In terms of the resource allocation problem, TBSTs quantify and anticipate uncertainties in demand predictions made by non-dispatchable agents. In particular, TBSTs allow an autonomous system to deal with volatile demand that follows different behavioral patterns. In power systems, this is the case with the output of weather-dependent power plants and the load of non-dispatchable consumers. In such situations, it would be risky to rely on a single expectation of how the demand will develop. Each TBST therefore consists of multiple scenarios, each representing a possible development of the demand and having a certain probability of occurrence. TBSTs lay the foundation for creating robust solutions to the scheduling problem on the basis of techniques from the field of online stochastic optimization. Trust-Based Scenario Trees are presented in Chapter 5.

Specification of Necessary Characteristics of Self-Organizing Hierarchies The scheduling problem is NP-hard with regard to the number of participating agents. To be able to solve this problem in large-scale open systems, we introduce a combination of techniques that allow for self-organized hierarchical problem decomposition. One way to implement a self-organizing hierarchical structure is to recursively solve the so-called *partitioning problem*, i.e., to partition a set of agents into disjoint

subsystems. Each of these subsystems represents a smaller, tractable sub-problem of the overall scheduling problem. We demonstrate that the following characteristics allow the system to autonomously come to a compromise between stability, scheduling times, and solution quality in terms of demand satisfaction and costs (see Section 9.2):

- **Constraints Guiding the Self-Organized Problem Decomposition** The size and the number of created subsystems – as well as the height of the resulting hierarchy – are important factors influencing scheduling times and solution quality. We present *partitioning constraints* that steer the formation of hierarchical system structures by means of suitable ranges for the number and the size of partitions. We introduce these constraints in Section 6.2.
- **An Organizational Paradigm Promoting the System’s Stability, Robustness, and Scalability** The class of open systems regarded in this thesis has to deal with uncertainties originating from participants that cannot or should not be excluded from the system. We present a new organizational paradigm, called *homogeneous partitioning*, that yields a structure of similar organizations. That way, each organization is equally able to cope with disturbances originating from non-dispatchable agents. Furthermore, the complexity of solving the scheduling problem is equalized among the organizations, which is beneficial to the overall runtime performance. Essentially, homogeneous partitioning increases the hierarchy’s stability and promotes the system’s robustness in terms of its ability to hold the balance between supply and demand. The idea of homogeneous partitioning is explained in Section 6.3.

Algorithms for the Self-Organized Formation of Hierarchical System Structures Finding the optimal solution to the partitioning problem requires solving an NP-hard combinatorial optimization problem. We present two self-organization algorithms, called *PSOPP* and *SPADA*, that solve the partitioning problem in a general manner, i.e., independently of the characteristics of a specific objective function. This property allows a system to create adequate partitionings with regard to a combination of different formation criteria. Self-organizing hierarchies are obtained in combination with a control loop (see Section 9.1) that uses PSOPP and SPADA to recursively solve instances of the partitioning problem. PSOPP and SPADA are ideally suited for the self-organized formation of hierarchies because they create high-quality partitionings that comply with the partitioning constraints. While PSOPP is especially suited for finding high-quality solutions for low-dimensional objective functions, SPADA’s strengths lie in multi-objective contexts:

- **A Discrete Particle Swarm Optimizer** Chapter 7 outlines self-organization algorithm PSOPP. PSOPP is based on particle swarm optimization – a computational method and population-based metaheuristic for optimization in large search spaces. To be independent of the characteristics of a specific objective function, PSOPP relies on basic set operations to come to a solution.
- **A Decentralized Agent-Based Partitioning Algorithm** SPADA deals with the complexity of the partitioning problem by implementing a decentralized agent-based approach instead of a (centralized) population-based metaheuristic. In SPADA, the group of agents that is to be partitioned uses an overlay network to autonomously decompose the overall partitioning problem into overlapping sub-problems. These are optimized in an iterative manner. Because the sub-problems overlap and change from one iteration to another, their beneficial characteristics spread out across other parts of a candidate solution, which ultimately leads to a high-quality overall partitioning. Despite its decentralized nature, SPADA guarantees compliance with the partitioning constraints. SPADA is introduced in Chapter 8.

Concepts and Algorithms for Robust Resource Allocation in Self-Organizing Hierarchies

Based on our concept of Trust-Based Scenario Trees (TBSTs), we introduce techniques for finding robust solutions to the resource allocation problem. Self-organizing hierarchies are prerequisite to deal with the complexity of the scheduling problem and the uncertainties involved. In this context, the contributions of this thesis are twofold:

- **A Specification of Robust Solutions to the Scheduling Problem** We combine techniques from the field of online stochastic optimization with our concept of TBSTs and apply them to

hierarchical schedule creation. TBST-based schedules thus enable the system to deal with different possible developments of the demand. To promote the agents' ability to reactively adapt to unforeseen situations, we also allow them to schedule an appropriate amount of degrees of freedom, i.e., reserves. The creation of TBST-based schedules and the provision of reserves are explained in Chapter 11. Chapter 12 shows that TBST-based schedules proactively guide the dispatchable agents' permanent reactive supply adjustments that are necessary to meet the demand at all times.

- **An Algorithm for the Scalable Creation of Robust Schedules** To find high-quality robust solutions to the scheduling problem in large-scale systems, we present an auction- and trust-based mechanism for hierarchical systems, called TruCAOS. TruCAOS reduces the problem's complexity by enabling the dispatchable agents to actively participate in the process of schedule creation. Consequently, the agents can self-organize into larger subsystems, which is advantageous to the schedules' quality and the system's stability. Among other trust-based measures, TruCAOS actively reduces uncertainties stemming from dispatchable agents that struggle with adhering to their schedules by "moving" their supply into regions in which their behavior is more predictable.

Thorough Evaluations in the Decentralized Power Management Case Study All concepts, approaches, and algorithms devised in the context of this thesis have been analyzed in extensive empirical evaluations that confirm their applicability, suitability, and necessity. The evaluations have been performed in an elaborate simulation environment for autonomous power systems on the basis of real world data. This simulation environment has been developed in the course of this dissertation in cooperation with other members of the working group.

Chapter 14 summarizes the research contributions and the most important evaluation results. In Chapter 15, we point to open research challenges and future directions.

Part II

Computational Trust as Enabler for Self-Organized Robust Resource Allocation

A major challenge of the resource allocation problem regarded in this thesis is to hold the balance between supply and demand despite uncertainties in the form of deviations from predicted demand or scheduled supply. If prior deviations from predictions and schedules are indicative of future behavior, agents can use their observations to deduce a probabilistic model of their interaction partners. In this thesis, we use the social concept of *trust* as a metaphor for measuring the accuracy of an agent's demand predictions or its compliance with created schedules. We define an agent's trustworthiness to be the higher, the lower and the less varying its deviations are.

In this part, we propose different trust models as enabler for self-organized robust resource allocation. In Chapter 4, we give an overview of computational trust based on the body of literature, first. Afterwards, we define a trust model to quantify and anticipate uncertain demand and supply. Among other things, this model serves as the basis for the self-organized formation of stable hierarchical system structures that promote the system's robustness. In Chapter 5, we outline the concept of *Trust-Based Scenario Trees* (TBSTs) for situations in which agents show volatile behavior that follows different behavioral patterns, and that require to capture and anticipate dependencies in a sequence of observed deviations. As opposed to common trust models that record an agent's average behavior or its variation, TBSTs model an agent's behavior by means of a set of scenarios. Each scenario has a certain probability of occurrence, and indicates in which way the behavior might develop in a sequence of future time steps. We propose to use TBSTs to quantify and anticipate uncertainties in demand predictions.

Trust as a Measure of Uncertainty

Summary. The resource allocation problem considered in this thesis has to be solved in a highly uncertain environment in which the agents' benevolence cannot be assumed. For this reason, uncertainties not only have to be anticipated but also quantified in order to allow the system to operate more robustly and efficiently. Since uncertainties are subject to change over time, they must be assessed at runtime. One way to address this challenge is to use the concept of *computational trust*. By extending the notion of trust as a qualifier of relationships between agents and incorporating trust into the agents' decisions, they can cope with uncertainties stemming from unintentional as well as intentional misbehavior. Based on the body of literature, we survey the most important properties of trust. Afterwards, we provide a basic trust model to quantify and anticipate uncertain demand and supply. This model is used to create adequate hierarchical system structures on the one hand, and serves as the basis for other trust models which we apply in the context of autonomous schedule creation on the other hand.

Publication. The findings described in this chapter have been published in Anders et al. [13, 14].

As explained in Chapter 1, inertia requires the intermediaries to allocate resources proactively on the basis of demand predictions requested from their subordinate non-dispatchable agents. However, because the demand is based on a stochastic process driven by the environment, predictions might turn out to be wrong, resulting in deviations between the actual and the predicted demand. Furthermore, dispatchable agents might not be able to contribute as stipulated in their schedule. In electric power systems, such deviations stem from inaccurate or outdated standard load profiles used to predict the load, the prosumers' geographic location influencing local weather conditions, imprecise sensor data used to predict future output, and technical difficulties, among others (see, e.g., [81, 86]).

One way to cope with these uncertainties is to create a probabilistic model that captures the underlying stochastic process in order to (1) anticipate *aleatoric*, i.e., intrinsic random and irreducible, uncertainties, and (2) reduce *epistemic*, i.e., systematic, uncertainties in the agents' behavior. Because an agent's behavior can vary and even completely change over time, e.g., as a result of changing environmental conditions, a precise offline model of the stochastic process is of little use. Consequently, such models have to be learned online and incorporate up-to-date information about the agents' behavior.

In computer science, *computational trust* (cf. [150]) has been proposed to allow agents to quantify and anticipate uncertainties that originate from the behavior of their interaction partners (e.g., a typical inaccuracy in a consumer's prediction due to the use of an outdated standard load profile). An agent's trust in another agent is based on *experiences* gained in past interactions. Similarly to [101, 226], we regard the creation of trust relationships as deducing a probabilistic model of an interaction partner at runtime (i.e., a form of *statistical inference* and *statistical learning*). Trust can be applied if it is valid to assume that an agent's prior behavior is indicative of its future behavior. As a consequence, trust serves as a predictive model that allows agents to make evidence-based assumptions about their interaction partners' future behavior.

Section 4.1 gives an overview of important properties of computational trust based on the body of literature and describes how trust evolves over time. In Section 4.2, we define a basic trust model to quantify and anticipate uncertain demand and supply when solving our resource allocation problem. Throughout this thesis, we use this model as the basis for the self-organized formation of stable hierarchical system structures that promote the system’s robustness in terms of its ability to hold the balance between supply and demand (see Chapter 6). Furthermore, this trust model serves as a basis for more specialized trust models that we introduce and employ in the context of an auction-based scheduling mechanism (see Chapter 10).

4.1 Properties of Trust

The classic notion of *computational trust* in the MAS community is focused on the credibility of agents, i.e., the degree to which they fulfill their commitments. This view stems mainly from psychological and sociological research [39] and boils down to the selection of interaction partners in order to maximize the utility of individual interactions. Economic [25, 177] and computer science [55, 150] literature characterize trust as an instrument to manage *expectations* about others. In computer science, the term “computational trust” is used to stress that the trust in a system or a system’s part, e.g., an agent, is assessed by means of a well-defined metric. Since both (a part of) the system or a human being can act in the role of the trustor, we can differentiate between system-to-system and user-to-system trust. In this thesis, we only regard trust between autonomous software agents. Often, a strong connection between trust and risk is emphasized [122] since interactions that incur a high risk for the participating agents require a high expectation of the others’ willingness to contribute in a beneficial manner. An empirically justified expectation reduces the *uncertainty* about the behavior of another agent [171].

In computing systems, trust is usually measured as a numerical *trust value*, often normalized to values between 0 and 1 [141]. In [142], an agent’s trust value is either very high or very low depending on whether the agent is either always expected to behave beneficially or never; if the value is between these extremes, the agent behaves in an unpredictable fashion and thus interactions with it are afflicted with a high uncertainty. Such a simple representation of trust is used in many trust models. However, there are numerous other interpretations and representations of trust (for an overview, see, e.g., [233]). Other representations based on more complex data structures (e.g., [181, 226]) are able to capture further properties, such as the volatility of an agent’s behavior. Before discussing the general properties of trust, we illustrate the life-cycle of trust values which can be transferred to most representations of trust, including those presented in this thesis.

The Life-Cycle of Trust Values. There is a general way of thinking about the origin of trust values that is independent of the way they are used (see Figure 4.1). Two or more parties commit to a (potentially implicit) *contract* [172] that defines an *interaction* (possibly composed of several distinct steps) as well as its *stipulated result*. The *actual result* of the interaction can be compared to what was stipulated in the contract, thus yielding an *experience* for each party [100]. Ultimately, an agent uses its experiences and a trust metric to derive a *trust value* for each of its interaction partners. Trust values, in turn, provide important information for future interactions. In Section 4.2, we instantiate this life-cycle for the resource allocation problem regarded in this thesis.

Falcone and Castelfranchi [66] criticized that many trust models are void of semantics of how the generated trust values have to be interpreted. It is, e.g., often not defined what a trust value of, say, 0.5 actually expresses or which trust value should be assigned to a new agent (the problem of *initial trust*, see, e.g., [144]). If a trust model has precise semantics, meaning a clearly defined way to interpret generated trust values, an abstracting numerical quantification can still be valid, though.

Properties of Trust. The life-cycle shows why trust values are usually *subjective*. As each agent makes its own experiences with others, it forms a “personal” opinion (i.e., a trust value) based on these unique experiences. Thus, the experiences of two agents with the same partner can vary tremendously. Additionally, agents can use different metrics to assess trust values and apply different requirements to the behavior of others, thus implementing different trust models. The same arguments can be used to argue against *transitivity* of trust [102]. An exception are recommendations as a form of *indirect trust*

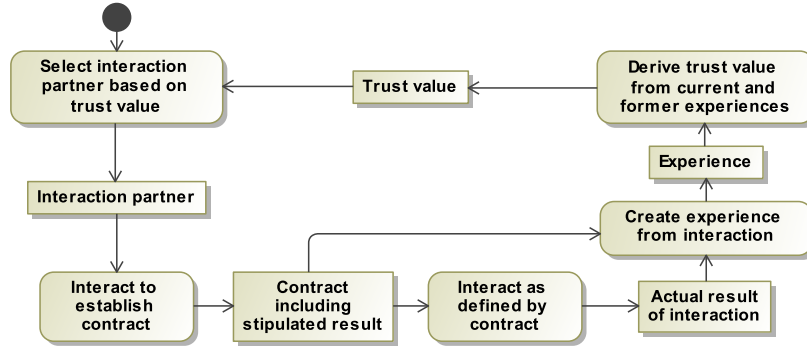


Figure 4.1: The life-cycle of trust values derived from experiences (adapted from [204]).

or *reputation* (see discussion below) that have to be based on a mutual understanding of the valuation of an agent’s behavior.

Further, it is crucial to consider the *context* in which interactions occur. The context includes, e.g., the roles the agents play in the interaction, its contract, or environmental circumstances. In the power management case study, a useful trust context might be, e.g., the time of day an output or load prediction is made for. Comparing experiences to each other in different contexts is difficult: A solar power plant is likely to make much more accurate output predictions for the night than for the day. Falcone and Castelfranchi [66] relate to context when they mention the “competence belief” an agent has about another. Competence is specific to a certain goal that the trusting agent believes the potential partner is capable to pursue. Clearly, agents that are deemed competent for one goal can be incompetent for another. Other authors use, e.g., “circumstance” [74] or “domain of interaction” [99] to denote context.

A trust value can also be supplemented by a measure of *confidence* [112] or *certainty* [82, 226] that indicates the degree of certainty that a trust value describes the actual observable behavior of an agent. Such an additional value can be based on several criteria, such as how many experiences were used for the calculation of the trust value, how old these experiences are, or how much the experiences differed. It is also possible to take the social relationships between the agents into account [127] or to distinguish short-term and long-term behavior in order to identify changing behavior.

Reputation. In open self-organizing systems, interaction partners can often change, e.g., due to alterations in system structure or inclusion of new agents. Since the agents’ benevolence cannot be assumed, they might not be willing to communicate their true intentions [190]. To deal with this situation, a reputation system can be used which combines the opinions of agents and generates recommendations [150]. This enables cooperation between agents that do not know or have only little experience with each other. To make adequate decisions, agents can rely on a combination of direct trust and reputation [113, 181]. Due to the subjective nature of trust and because agents might lie about the trustworthiness of others, it is often also desirable to weigh the impact a recommending agent (called *witness*) has on the reputation value [20, 111]. Whenever a reputation system is used, there has to be a consensus among the agents about the meaning of trust and reputation values. A common trust model can fulfill this purpose. In this thesis, we do not further distinguish between direct trust and reputation. We assume that agents truthfully provide others with information about another agent’s trustworthiness if requested.

There is a variety of different uses of trust in open self-organizing systems, including trust to inform self-organization processes, to optimize for critical or likely situations in uncertain environments, to reduce uncertainties resulting from intentional misbehavior by means of appropriate sanctioning or incentive mechanisms, and to represent the social relationships of the system’s users. In this thesis, we concentrate on (1) the trust-based and self-organized structuring of large-scale open systems, and (2) trust-based schedule creation as a means to scalable and robust resource allocation under uncertainty.

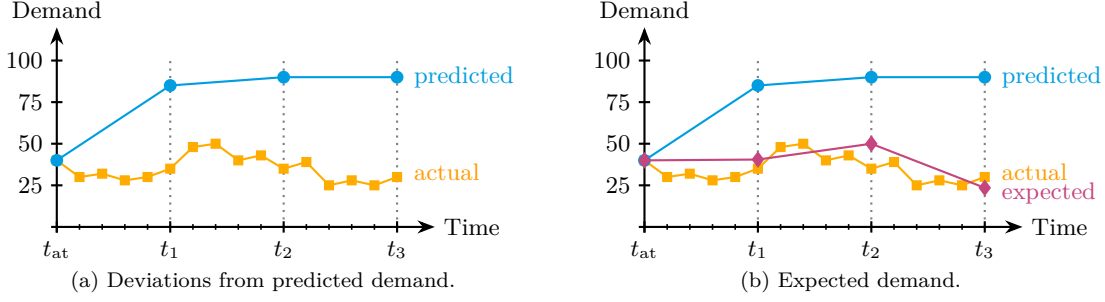


Figure 4.2: Example of an inaccurate demand prediction: The dotted vertical lines indicate the time steps $t_i = t_{at} + i \cdot \Delta\tau$ that constitute the scheduling window \mathcal{W} in time step t_{at} . As illustrated in Figure 4.2a, uncertainties manifest as deviations between the actual $\langle 35, 35, 30 \rangle$ and the predicted demand $\langle 85, 90, 90 \rangle$. With regard to t_1 , for instance, the intermediary gains an atomic experience $E_{t_{at}}[t_1] = (35, 85, t_{at}, t_1)$ that captures the deviation $\delta(E_{t_{at}}[t_1]) = 35 - 85 = -50$. The corresponding sequential experience $E_{t_{at}}$ records the sequence of deviations $\langle -50, -55, -60 \rangle$. Figure 4.2b shows that the influence of inaccurate predictions can be mitigated by relying on expectations instead of the prediction itself. An intermediary derives an agent's expected demand from its demand prediction and the trust value reflecting its expected deviation.

These uses are addressed in Chapter 6 and Part IV. There, we also discuss more problem-oriented related work from the field of computational trust and related research areas.

4.2 Quantifying and Anticipating Uncertainties in Predicted Demand and Scheduled Supply by Means of Trust Values

As discussed at the beginning of this chapter, we use the concept of computational trust to quantify and anticipate uncertainties in predicted demand and scheduled supply at runtime in order to enable self-organized robust resource allocation. More precisely, the agents utilize the information about the trustworthiness of others in the self-organized formation of scalable system structures and when creating schedules. Both uses aim at anticipating uncertainties (i.e., deviations from predicted demand or scheduled supply) to mitigate their impact on the system. In Chapter 10, we will show that a trust-based scheduling mechanism can incentivize benevolent behavior and actively reduce uncertainties by putting dispatchable agents into states of less uncertainty.

With regard to the hierarchical system structure, we only regard trust relationships describing an intermediary's trust in its subordinate agents. We assume that intermediaries truthfully provide other agents with information about the trustworthiness of their subordinates if needed, such as during the self-organized formation of the hierarchy. The trust relationships have their origin in contracts created in the course of the schedule creation. We regard two different types of contracts: The first type comprises the schedules of dispatchable agents. The second type encompasses the demand predictions of non-dispatchable agents whose behavior usually depends on environmental conditions, such as the solar radiation in case of a solar power plant. Each contract is concluded between an intermediary (acting in the role of the trustor) and one of its subordinate agents (acting in the role of the trustee).

Recall that, with regard to a specific time step t_{at} , schedules are created for the scheduling window $\mathcal{W} = \{t_{at} + i \cdot \Delta\tau \leq t_{at} + H \mid i \in \mathbb{N}_{\geq 1}\}$ that comprises $N = H/\Delta\tau$ time steps (see Section 1.2). We therefore define a contract $C_{t_{at}} = \langle C_{t_{at}}[t_{at} + \Delta\tau], \dots, C_{t_{at}}[t_{at} + H] \rangle$ representing a demand prediction (resp. a schedule) as a sequence of N predicted demand (resp. scheduled supply) values $C_{t_{at}}[t_{for}]$ with $t_{for} \in \mathcal{W}$. While t_{at} stands for the time step in which the contract was concluded, t_{for} denotes the time step in which the agent should behave as stipulated in $C_{t_{at}}[t_{for}]$.

Given a predicted demand (scheduled supply) $C_{t_{at}}[t_{for}]$, an intermediary gains an *atomic experience* $E_{t_{at}}[t_{for}] = (A[t_{for}], C_{t_{at}}[t_{for}], t_{at}, t_{for})$ by measuring the actual demand (supply) $A[t_{for}]$ at time

step t_{for} .¹ The experience captures the deviation $\delta(E_{t_{\text{at}}}[t_{\text{for}}]) = A[t_{\text{for}}] - C_{t_{\text{at}}}[t_{\text{for}}]$ between the actual demand (supply) and the stipulated value. With regard to the entire contract $C_{t_{\text{at}}}$, an intermediary gains a *sequential experience* $E_{t_{\text{at}}} = \langle E_{t_{\text{at}}}[t_{\text{at}} + \Delta\tau], \dots, E_{t_{\text{at}}}[t_{\text{at}} + H] \rangle$, that is, a sequence of atomic experiences (in the following, we only distinguish between atomic and sequential experiences where needed). Figure 4.2a shows an example of a demand prediction in comparison to the actually measured demand. In the following, we explain how intermediaries use such experiences to assess the trustworthiness of their subordinate agents.

A Basic Trust Model to Quantify and Anticipate Uncertain Demand and Supply

To cope with uncertain demand and supply, each intermediary assesses the behavior of each of its subordinate agents by means of two trust values: The first trust value records an agent's *mean* deviation from its predicted demand or scheduled supply and thus captures *systematic* misbehavior. In turn, this value serves as an expectation of an agent's deviation from its predictions or schedules.² However, when calculating this trust value, positive and negative deviations can cancel each other out, e.g., leading to an expected deviation of 0 even though an agent consistently yields extremal deviations. The second trust value is therefore used to mirror an agent's *predictability* in terms of the dispersion of its deviations. It represents the risk that the goal of balancing supply and demand is not achieved because the agent does not consume or provide resources as stipulated or expected. The predictability is thus an indicator of aleatoric uncertainties: the less aleatoric uncertainties, the more predictable is an agent's behavior.

Assume that an intermediary λ holds the $n_{\mathcal{E}}$ newest atomic experiences with one of its subordinate agents $a \in \mathcal{A}_{\lambda}$ in a set \mathcal{E} . We only regard the $n_{\mathcal{E}}$ newest experiences because an agent's behavior may change over time (e.g., due to temporary environmental influences, such as snow on a solar panel) and we assume that more recent behavior can give a better indication of an agent's future behavior. In case of former negative experiences, this also allows for *forgiveness* [219]. Equally, prior positive experiences lose their relevance over time if the corresponding agent now notoriously violates its contracts. Note that this is similar to the calculation of a moving average with a sliding window of size $n_{\mathcal{E}}$. From these experiences \mathcal{E} (with $|\mathcal{E}| \leq n_{\mathcal{E}}$), the intermediary λ derives a trust value $T_{\mu}(\mathcal{E})$ that assesses how much it trusts agent a to adhere to its contract. We define $T_{\mu}(\mathcal{E})$ as a 's mean deviation from its contracts:

$$T_{\mu}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \cdot \sum_{E_t[t'] \in \mathcal{E}} \mathfrak{N}(E_t[t']), \quad (4.1a)$$

$$\text{with } \mathfrak{N}(E_t[t']) = \frac{\delta(E_t[t'])}{\delta^{\max}} \quad (4.1b)$$

The function $\mathfrak{N}(E_t[t'])$ normalizes the deviation $\delta(E_t[t'])$ to an interval $[-1, 1]$ by means of the normalization factor $\delta^{\max} > 0$ which, e.g., represents the maximal absolute deviation seen in \mathcal{E} . The resulting trust value $T_{\mu}(\mathcal{E}) \in [-1, 1]$ represents a 's *expected deviation* $\mathbb{E}(\delta) = T_{\mu}(\mathcal{E}) \cdot \delta^{\max}$ from a predicted demand or scheduled supply. The greater $|T_{\mu}(\mathcal{E})|$, the greater the absolute value of the expected deviation and thus the less trustworthy a . Note that we do not prefer positive to negative deviations or vice versa. In the power management case study, both an over- as well as an underestimated residual load or supply negatively affect the power grid's utility frequency. This is in contrast to other trust models, such as [172], who regard positive deviations as highly desirable outcomes. If $T_{\mu}(\mathcal{E}) < 0$, λ expects that a 's actual demand (supply) will be lower than stipulated. If $T_{\mu}(\mathcal{E}) > 0$, λ expects that the actual demand (supply) is greater than stipulated. Otherwise, if $T_{\mu}(\mathcal{E}) = 0$, λ expects a to comply with its contract. As also the magnitude of deviations is captured in $T_{\mu}(\mathcal{E})$, this trust metric is semantically sound (cf. our discussion in Section 4.1).

Because the trust value $T_{\mu}(\mathcal{E})$ reflects the *mean* deviation from contracts, positive and negative experiences can cancel each other out, i.e., $T_{\mu}(\mathcal{E})$ might be 0 although a always deviates from its

¹We assume that each intermediary is able to measure the actual supply or demand of its subordinate agents.

²Note that we differentiate between *expected* / *anticipated* behavior and *predicted* behavior. While predictions are made by a trustee, the trustor uses a trust model (which serves as a predictive model in terms of machine learning) to anticipate the trustee's deviating behavior by deriving an expected behavior from prior experiences.

contracts. Since an agent's *predictability* decreases with the volatility of its behavior, we use the standard deviation $T_v(\mathcal{E}) \in [0, 1]$ of the normalized deviations in \mathcal{E} as a metric. The concept of predictability thus corresponds to the criteria “variance of experiences” of the *confidence metric* we introduced in [112]. The higher an agent's predictability, the more its intermediary can rely on the expected deviation derived from the first trust value. Expected variations in contract compliance are therefore defined as $v(\delta) = T_v(\mathcal{E}) \cdot \delta^{\max}$.

Given a trust value $T_\mu(\mathcal{E})$, an intermediary λ can anticipate a 's actual demand or supply on the basis of its expected deviation and a contract. For instance, if a provides λ with a demand prediction $\widehat{D}_{t_{\text{now}}}^\mathcal{W} = \langle \widehat{D}_{t_{\text{now}}}^\mathcal{W}[t_{\text{now}} + \Delta\tau], \dots, \widehat{D}_{t_{\text{now}}}^\mathcal{W}[t_{\text{now}} + H] \rangle$, λ can use a 's expected deviation $\mathbb{E}(\delta)$ to derive an *expected* demand for a series of future time steps. We define the corresponding expected demand $\overline{D}_{t_{\text{now}}}^\mathcal{W} = \langle \overline{D}_{t_{\text{now}}}^\mathcal{W}[t_{\text{now}} + \Delta\tau], \dots, \overline{D}_{t_{\text{now}}}^\mathcal{W}[t_{\text{now}} + H] \rangle$ by the sum of the stipulated value and the expected deviation such that $\overline{D}_{t_{\text{now}}}^\mathcal{W}[t_{\text{for}}] = \widehat{D}_{t_{\text{now}}}^\mathcal{W}[t_{\text{for}}] + \mathbb{E}(\delta)$ for each $t_{\text{for}} \in \{t_{\text{now}} + \Delta\tau, \dots, t_{\text{now}} + H\}$. Figure 4.2b shows that an intermediary can significantly reduce the impact of inaccurate predictions by relying on expectations instead of predictions.

Note that we opted for creating a predictive model of an agent's behavior in terms of its deviations instead of its actual demand or supply. This is because some non-dispatchable agents make better predictions about their future demand than others. Further, anticipating the agents' actual demand without incorporating their prediction requires additional information, which makes the task of creating an adequate probabilistic model much more difficult. With regard to dispatchable agents, this is mainly due to the ability to influence the agent's behavior by means of a schedule. In the following, we extend this basic trust model by a trust context that allows for an improved anticipation of uncertain demand.

Using the Prediction Horizon as Trust Context to Anticipate Uncertain Demand

As stated in Section 1.3, we assume that demand predictions become more accurate as the point in time they are made for approaches (note that we do not make this assumption for scheduled supply). The basic trust model sketched above does not take account of this property: We derive the expected demand by adding the expected deviation $\mathbb{E}(\delta)$ as a constant offset to the prediction. In fact, the expected demand shown in Figure 4.2b cannot stem from our basic trust model because the difference between predicted and expected demand varies with the prediction horizon. To improve the accuracy of expected deviations, we now extend our basic trust model by a trust context that allows an intermediary λ to quantify the accuracy of demand predictions with respect to the temporal distance to a future point in time. In other words, we employ the prediction horizon as trust context and group experiences according to the temporal distance between t_{at} and t_{for} .

As illustrated in Figure 4.3, an intermediary gains multiple experiences per time step. That is because each time schedules are created, the demand is predicted for the N time steps of the scheduling window $\mathcal{W} = \{t_{\text{at}} + i \cdot \Delta\tau \leq t_{\text{at}} + H \mid i \in \mathbb{N}_{\geq 1}\}$. If we assume that schedules are recalculated after the time span $\Delta\tau$, there are exactly N demand predictions for every time step t_{for} that matches the schedule time pattern (recall that t_{for} is thus a multiple of the schedule resolution $\Delta\tau$). These predictions were made in the N previous time steps $t_{\text{at}} \in \{t_{\text{for}} - i \cdot \Delta\tau \geq t_{\text{for}} - H \mid i \in \mathbb{N}_{\geq 1}\}$. As a result, each intermediary gains N experiences at once in each time step t_{for} . As before, an experience $\delta(E_{t_{\text{at}}}[t_{\text{for}}]) = A[t_{\text{for}}] - \widehat{D}_{t_{\text{at}}}^\mathcal{W}[t_{\text{for}}]$ captures the difference between a non-dispatchable agent's actual $A[t_{\text{for}}]$ and predicted demand $\widehat{D}_{t_{\text{at}}}^\mathcal{W}[t_{\text{for}}]$.

Due to our assumption that predictions become more accurate as t_{for} approaches, the accuracy of a predicted demand $\widehat{D}_{t_{\text{at}}}^\mathcal{W}[t_{\text{for}}]$ depends on the prediction horizon, i.e., the temporal distance, $\Delta h \in \{i \cdot \Delta\tau \leq H \mid i \in \mathbb{N}_{\geq 1}\}$ between the time step $t_{\text{at}} \in \{t_{\text{for}} - i \cdot \Delta\tau \geq t_{\text{for}} - H \mid i \in \mathbb{N}_{\geq 1}\}$ and t_{for} . To capture this behavior, each intermediary manages N sequences of atomic experiences $\mathcal{E}_D^{\Delta h} = \langle E_{t_{n_D} - \Delta h}[t_{n_D}], \dots, E_{t_1 - \Delta h}[t_1] \rangle$ (with $t_i = t_{\text{now}} - (i - 1) \cdot \Delta\tau$), one for each distance $\Delta h \in \{i \cdot \Delta\tau \leq H \mid i \in \mathbb{N}_{\geq 1}\}$.³ That way, $\mathcal{E}_D^{\Delta h}$ contains the n_D newest experiences $E_{t_{\text{at}}}[t_{\text{for}}]$ with $\Delta h = t_{\text{for}} - t_{\text{at}}$. With regard to Figure 4.3 and time step 10:45, an intermediary holds the sequence of experiences $\mathcal{E}_D^{15 \text{ min}} = \langle E_{10:00}[10:15], E_{10:15}[10:30], E_{10:30}[10:45] \rangle$ for the prediction horizon $\Delta h = 15 \text{ min}$ and a

³The index “D” in $\mathcal{E}_D^{\Delta h}$ and n_D indicates that we regard uncertain demand.

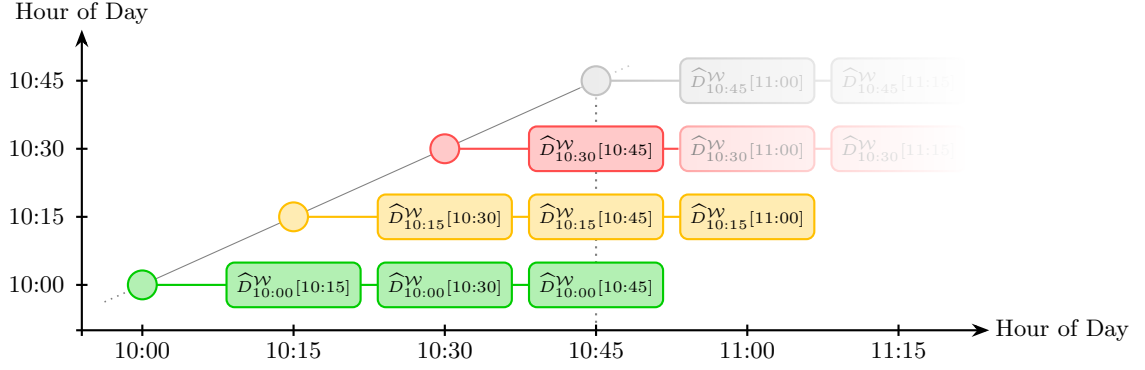


Figure 4.3: Four demand predictions made for $N = 3$ consecutive time steps and a schedule resolution $\Delta\tau = 15$ min at four different points in time $t_{at} \in \{10:00, 10:15, 10:30, 10:45\}$. The figure illustrates the relation between a demand prediction $\hat{D}_{t_{at}}^W[t_{for}]$ for a specific time step t_{for} and the temporal distance $\Delta h = t_{for} - t_{at}$ (in this example, $\Delta h \in \{15 \text{ min}, 30 \text{ min}, 45 \text{ min}\}$). As indicated at 10:45, the corresponding intermediary gains $N = 3$ experiences in each time step.

maximum of $n_D = 3$ experiences. Visually, sequences $\mathcal{E}_D^{\Delta h}$ can be thought of as the “secondary diagonals” in Figure 4.3.

To determine a non-dispatchable agent’s trust value for the prediction horizon $\Delta h = t_{for} - t_{now}$, we reuse the function T_μ (see Equation (4.1a)). This time, however, we evaluate the set of experiences $\mathcal{E}_D^{\Delta h}$ instead of \mathcal{E} . Hence, the trust context serves as a filter for experiences. As before, $T_\mu(\mathcal{E}_D^{\Delta h}) \in [-1, 1]$ represents an expected deviation $\mathbb{E}(\delta) = T_\mu(\mathcal{E}_D^{\Delta h}) \cdot \delta^{\max}$ from a predicted demand $\hat{D}_{t_{now}}^W[t_{for}]$, but this time specifically for the trust context Δh .

Similarly, it is possible to introduce a trust context that evaluates an agent’s behavior with regard to the time of day for which a contract is concluded. For instance, such a trust context might be beneficial to assess the accuracy of output predictions of solar power plants separately for daytime and nighttime.

Chapter Summary and Outlook

In this chapter, we reviewed the properties of computational trust, and devised a basic trust model that allows the agents to quantify and anticipate uncertainties in predicted demand and scheduled supply. This trust model regards two dimensions of trust: The first dimension captures an agent’s systematic misbehavior in the form of a trust value that reflects its mean deviation from its predicted demand (in case of a non-dispatchable agent) or its scheduled supply (in case of a dispatchable agent). In the second dimension, another trust value measures an agent’s predictability in the form of the standard deviation of its deviations.

In Chapter 10, we show that both dimensions of trust play an important role in the creation of schedules that yield an adequate balance between supply and demand. In this context, the basic trust model serves as a basis for more specialized trust models that we employ in an auction-based scheduling mechanism. In the course of the self-organized formation of hierarchical system structures (see Part III), an agent’s trustworthiness is mainly regarded in terms of its predictability. To create a structure in which each organization is equally able to compensate for local deviations between supply and demand, we aim for a similar ratio between uncertainty (i.e., predictability) and degrees of freedom in terms of controllability.

The basic trust model presented in this chapter cannot anticipate agent behavior that follows different behavioral patterns or exhibits sequential dependencies in a series of observations, though. In the next chapter, we introduce a trust model called *Trust-Based Scenario Trees* which applies to such situations.

Trust-Based Scenario Trees to Anticipate Volatile Demand that Follows Different Behavioral Patterns

Summary. In some situations, it is necessary to capture time-dependent behavior in the sense that the quality of a prediction depends on the prediction error in previous time steps. As for power systems, this is the case with weather-dependent power plants and non-dispatchable consumers. Furthermore, an expectation of how an agent will behave in the future can turn out to be wrong. This is especially the case if the agent's behavior is volatile, even if it follows different behavioral patterns. With regard to the system's efficiency and robustness, it would therefore be risky to rely on a single expectation of how the agent behaves in the future. To meet these challenges, we introduce the concept of *Trust-Based Scenario Trees* (TBSTs) as an approach to probabilistic model creation in this chapter. In terms of our resource allocation problem, we use TBSTs to quantify and anticipate uncertainties in demand predictions. TBSTs are created at runtime. Each TBST consists of multiple scenarios each of which represents a possible sequence of deviations from a demand prediction. In combination with a demand prediction, a TBST describes different expected developments of the demand. In Part IV, we show that TBSTs lay the foundation for creating robust schedules on the basis of techniques from the field of *online stochastic optimization* [83]. In the context of the power management case study, we demonstrate that TBSTs significantly improve the agents' ability to anticipate future agent behavior in comparison to simple trust values. Moreover, we show that a single scenario of a TBST already reaches the same quality of expectations as an optimized hidden Markov model. The advantage of considering an entire tree of interrelated scenarios becomes evident when allowing the agents to switch to the most suitable scenario at runtime.

Publication. The concepts and results outlined in this chapter have been published in Anders et al. [13].

In this chapter, we concentrate on the creation of probabilistic models of the behavior of non-dispatchable agents. As is the case with the trust model presented in Section 4.2, we want to anticipate an agent's behavior in terms of deviations from its demand predictions. Anticipating its actual demand would require a much more detailed model and information that might not be available to other agents. Further, we focus on non-dispatchable agents in this chapter because intermediaries can influence the behavior of dispatchable agents by means of appropriate schedules. To create a meaningful probabilistic model, we have to meet four major challenges:

- (C1) The agents' behavior can change over time (e.g., due to temporary environmental influences, such as snow on a solar panel), which is why the probabilistic model must be flexible enough to adapt

to variable behavior. Note that, within a specific time frame, variable behavior might be quite stable and thus predictable.

- (C2) Only few assumptions about an agent’s behavior can be made (e.g., weather conditions influence consumers and producers in different ways), which is why the employed method must be able to reflect different types of agent behavior (i.e., stochastic processes).
- (C3) An agent’s behavior might show sequential dependencies in the sense that an inaccurate prediction for a specific time step might depend on the prediction error of previous time steps. Such sequential dependencies have to be captured.
- (C4) An agent’s behavior might be volatile yet follow different behavioral patterns, i.e., scenarios. These scenarios have to be distinguished in order to be able to anticipate the agent’s behavior. In such situations, one might not be sure which of these scenarios occurs. Therefore, relying on a single expectation of an agent’s future behavior might have serious consequences for the overall system if this expectation turns out to be wrong.

The following example illustrates challenges (C3) and (C4): Let us regard a person that uses an electric vehicle to commute. The vehicle’s battery is charged over night and it stops charging as soon as the person drives to work, leading to a decrease in the household’s demand. Usually (in 75 % of the cases), the person leaves home at 8:00 a.m. From time to time (in 25 % of the cases), the person does, however, not leave home until 9:00 a.m. so that the household’s demand remains high for one more hour. If we assume that the software agent representing the household always predicts a demand according to the person’s usual behavior, the intermediary requesting these predictions observes two different behavioral patterns (i.e., scenarios) for the prediction error, both with a certain probability of occurrence. In the first scenario, predictions are very accurate, corresponding to a prediction error of 0 kW. In the second scenario, there is a deviation of 10 kW between the predicted and the actual demand between 8:00 a.m. and 9:00 a.m. (assuming that the battery is charged at $10 \frac{\text{kW}}{\text{h}}$), followed by a precise prediction of the demand for the time the person is at work.

As discussed in Chapter 4, trust models meet challenge (C1) since trust values originate from experiences gained at runtime. Due to challenges (C2), (C3), and (C4), a single trust value reflecting the mean deviation, multiple trust values for different prediction horizons, or even trust values in combination with the agent’s predictability (i.e., the variance of deviations) are often not sufficient to obtain an informative probabilistic model. In our example, these approaches have two drawbacks: With respect to challenge (C3), a trust model based on multiple *unrelated* trust values – each for a different prediction horizon – is not able to express the process’s sequential dependencies in the sense that, once the prediction error decreased because the person left home, predictions are accurate for the next couple of hours. Usually, the relationships between the atomic experiences contained in a sequential experience get lost when creating one or more trust values. As for challenge (C4), a trust value representing the mean prediction error of $2.5 \text{ kW} = 10 \text{ kW} \cdot 0.25 + 0 \text{ kW} \cdot 0.75$ for the time between 8:00 a.m. and 9:00 a.m. does neither reflect the household’s behavior in the first scenario nor in the second scenario (knowledge about both scenarios would be optimal). Combining the mean prediction error with the corresponding standard deviation is insufficient to describe the actual probability distribution.

Due to challenge (C1), a precise offline model of the agent’s behavior is either not accurate enough for momentary situations or computationally not feasible for fast reactions. In this chapter, we introduce a trust model called *Trust-Based Scenario Trees* (TBSTs) that meets the challenges listed above. As other approaches based on the social concept of trust, TBSTs are created on the basis of past experiences and assume that prior observations are indicative of future behavior. Each TBST approximates the observed stochastic process by learning an empirical probability mass function specifying a discrete probability distribution over multiple time steps. In contrast to trust models that capture an agent’s average behavior or its variation, a TBST holds multiple possible scenarios, each indicating an expectation, i.e., a possibility, of how an agent’s behavior might develop in a sequence of future time steps. Each of these *Trust-Based Scenarios* has a certain probability of occurrence. These characteristics allow TBSTs to deal with sequential dependencies and volatile agent behavior that follows different behavioral patterns (cf. challenges (C3) and (C4)). As opposed to the concept of *scenario trees* as known from the domain of operations research [198], TBSTs make only few assumptions about the underlying stochastic process.

Further, they have been developed with the purpose of being *learned online* by agents with possibly low computational power and on the basis of a relatively low number of experiences. In Chapter 11, we outline how intermediaries utilize TBSTs in combination with the principle of *stochastic programming* [83] to create robust schedules for dispatchable agents.

The remainder of this chapter is structured as follows: We discuss related work in Section 5.1 before we explain the generation of TBSTs in Section 5.2. In Section 5.3, we evaluate the accuracy of the probabilistic models obtained by TBSTs to those created by our basic trust model presented in Section 4.2. Moreover, we compare the performance of TBSTs to *hidden Markov models* [38] – a well-known stochastic model from the literature that is used in various applications to approximate the model of a system whose states can only be partially observed.

5.1 Related Work

As mentioned in Section 4.1, some trust models propose to create more precise probabilistic models by means of additional measures that supplement the trust value describing an agent’s average behavior. Such supplementary measures are called, e.g., *certainty* [82, 226], *confidence* [112], or *reliability* [85, 97]. They have in common that they indicate the degree of certainty that the trust value mirrors an agent’s actual observable behavior. While all quantify an agent’s volatility, they are based on different metrics: He et al. [82] as well as the *confidence metric* we presented in [112] measure the standard deviation of experiences. Huynh et al. [97] and Hermoso et al. [85] use the mean absolute deviation as a measure of reliability. In [226], the mean absolute deviation is used to formulate a so-called probability-certainty density function, which is a probability density function of the probability of a positive experience. However, if we want to use these models to anticipate an agent’s behavior, we still have to make assumptions about its underlying probability distribution, i.e., the stochastic process. Moreover, the mean value and the standard deviation are insufficient to describe certain probability distributions. Consequently, these approaches do not comply with challenge (C2). Furthermore, they do not record sequential dependencies in an agent’s behavior (cf. challenge (C3)). Hence, they are not suitable for forecasting an agent’s behavior for time series. Although all approaches capture an agent’s volatility, they do not differentiate between different behavioral patterns (cf. challenge (C4)).

Li et al. [132] present a fuzzy regression-based approach to forecast uncertainties in the field of service-oriented applications. Forecasts are made on the basis of a trust value, a trust trend (i.e., an indication whether the trust value will rise or fall), and the performance consistency that reflects the trust value’s stability. While this model can forecast behavioral changes and trends in an e-service environment, the approach is not able to distinguish and anticipate multiple behavioral patterns (cf. challenge (C4)).

Hussain et al. [95] use a Markov model to forecast an agent’s trust value in the next time step. States mirror an agent’s trust value. The probability of state transitions is learned at runtime, depending on how often this transition was observed. With regard to the state that represents the agent’s current trust value, the forecasted trust value for the next time step is the state that is reached when choosing the most likely transition. While this model includes a basic mechanism to reflect sequential dependencies in behavior (the trust value in the next time step depends on the current trust value), it cannot capture sequential dependencies between more than two time steps (cf. challenge (C3)). This is due to the Markov property. Furthermore, it has not been devised to anticipate an agent’s behavior for more than a single time step and to anticipate different scenarios (cf. challenge (C4)).

Another trust model that is based on a Markov chain is introduced in [50]. In a mobile ad-hoc network, events, i.e., new experiences, trigger state changes in the Markov chain, i.e., changes in the trust value. While the Markov chain is used to analyze the trust model, it remains unclear how agents use information, such as transition probabilities, to make decisions or forecast future behavior.

Hidden Markov models (HMMs) generalize Markov chains by the ability to approximate the stochastic model of a system whose states can only be partially observed. For instance, while a drop in the demand of the commuter’s household can be observed, it is unknown if this is because the person left for work or for another reason. Similar to Markov chains, they often lack the ability to anticipate sequential

dependencies in behavior for multiple consecutive time steps (due to the Markov property) or cannot provide different (possibly related) scenarios of future behavior (cf. challenges (C3) and (C4)). However, HMMs have been successfully applied to temporal pattern recognition [68], which is also of interest in the context of modeling consumer behavior or the weather. For this reason, HMMs and Markov chains have been used to model the time-dependent output of wind turbines, e.g., in [152] and [235], respectively. Although HMMs do not meet all of our challenges, we use them for comparison in our evaluation mainly due to their ability to recognize different patterns and forecasting stochastic processes (see Section 5.3). Concerning the latter, HMMs have been applied to forecasting stock prices for interrelated markets [80], for example.

Hussain et al. [96] present several metrics that can forecast the trust value for the next time step despite (in)regularities in an agent’s behavior (such a behavior might be shown by strategic agents). However, the applicability of the metrics depends on the characteristic of the agent’s behavior and it is not possible to forecast different possible developments of an agent’s future behavior. These metrics basically exhibit similar drawbacks as the method presented in [95] (see discussion above).

Hernández et al. [86] use artificial neural networks to predict the load in electric power systems. Their approach is able to predict the load for a sequence of time steps, but requires a large amount of data to deliver accurate predictions. Basically, the approach could be extended to generate different scenarios of the load: Since it groups load curves with similar features into clusters, the mean of each cluster could represent a scenario. Probabilities are not provided, though (cf. challenge (C4)). As opposed to this approach, we do not want to predict the demand itself but anticipate an agent’s deviations from its demand predictions. As stated before, we opt for anticipating prediction errors, i.e., deviations, instead of the agents’ actual demand because some agents make better predictions about their future demand than others. Moreover, anticipating the agents’ demand without incorporating their prediction requires additional information, which makes the task of creating an adequate probabilistic model much more difficult.

In the domain of operations research and power systems, *scenarios* and *scenario trees* are proven concepts to describe a stochastic process, such as volatile wind power generation (see, e.g., [90, 235]). Each scenario represents a possible behavior over a series of future time steps and has a probability of occurrence. This directly addresses challenges (C3) and (C4). Because scenario trees discretize an agent’s behavior, they can serve as input for solving optimization problems under uncertainty, as is the case with *stochastic programming* [198]. Scenarios are usually generated on the basis of gathered historical data and expert knowledge [90]. A common method to create scenarios is called *moment matching*: Based on estimated moments of a (continuous) probability distribution, a number of scenarios has to be found that induce a discrete probability distribution that matches the given moments as well as possible [93]. Finding these scenarios requires solving an optimization problem. Due to the computational complexity of creating adequate scenarios and scenario trees, many approaches make assumptions that do not comply with the challenges sketched at the beginning of this chapter: Scenarios and their probabilities are often predetermined and generated offline or cannot mirror sequential dependencies [235]. Other methods predefine the tree’s structure, the number of scenarios, or the underlying probability distribution (cf. [41, 61]). In this context, Hochreiter and Pflug [90] distinguish between given structure problems (fixed structure of the tree), stage-wise fixed structure problems (fixed number of nodes per level), and free structure problems (fixed number of scenarios). When the underlying stochastic process is known, approaches based on Monte Carlo simulation can be used to generate a predefined number of scenarios (e.g., [225]). Other approaches, such as [78, 162], regard the problem of reducing the number of scenarios in a given tree to make it easier to handle, e.g., when solving an optimization problem on its basis. The challenge of scenario reduction is to remove scenarios while maintaining the model’s accuracy.

An approach for open self-organizing systems must not make such assumptions and has to be able to derive scenarios and their probabilities from up-to-date data at runtime. That is why computationally efficient solutions are needed.

5.2 Trust-Based Scenario Trees

Our concept of *Trust-Based Scenario Trees* (TBSTs) meets the challenges listed at the beginning of this chapter. With respect to the hierarchical system structure, each intermediary employs TBSTs to measure the accuracy of demand predictions of subordinate non-dispatchable agents and, in turn, to deduce their expected deviations. Recall that an intermediary's local demand also contains the local demand of subordinate intermediaries (see Section 2.2). As we will learn in Section 11.2, an intermediary intentionally does *not* capture uncertainties resulting from the local demand of subordinate intermediaries, though. That is because subordinate intermediaries are expected to deliver deviations of 0 by installing necessary precautions. A subordinate intermediary that cannot compensate for its local uncertainties is thus indicative of an improper system structure. In such a situation, a reorganization has to be triggered (see Part III). To derive adequate expected deviations for its local demand, an intermediary might have to capture statistical relationships between its non-dispatchable agents' behavior (e.g., think of weather predictions for adjacent regions or consumer attitude). To deal with this situation, an intermediary quantifies uncertainties originating from the *group* of subordinate non-dispatchable agents as a whole. In our case study, each AVPP creates a TBST to anticipate prediction errors of its local residual load that comprises its subordinate non-dispatchable power plants and, in case of the top-level AVPP, also the consumers.

Essentially, intermediaries create TBSTs on the basis of *sequential experiences* (i.e., sequences of deviations) gained in prior interactions with subordinate non-dispatchable agents. TBSTs have the structure of tree diagrams as known from probability theory. Similar to the creation of a histogram, a TBST is derived by classifying (i.e., discretizing) sequences of deviations by means of bins (i.e., intervals) of a predefined size. Each observed sequence of bins represents a *Trust-Based Scenario* (TBS). Together with an arbitrary root, a set of TBSs constitutes a TBST whose branches result from common prefixes of the TBSs. As TBSTs serve as predictive models, a TBS, in turn, stands for a possible sequence of deviations. The presumed probability that a TBS occurs depends on how often the corresponding sequence was observed relative to the occurrence of the other sequences. With regard to the tree structure, these probabilities are used to annotate a TBST's transitions with conditional probabilities that the expected deviation changes from one value to another. That way, an intermediary obtains an empirical probability mass function specifying a discrete probability distribution over multiple time steps. Every time schedules are created, intermediaries update their TBSTs with their latest experiences. As schedules are created for N time steps, the sequential experiences used for the generation of TBSTs encompass N deviations each. Consequently, a TBST can serve as predictive model for up to N future time steps. For the sake of clarity, we call such a TBST and its TBSs a *deviation tree* and *deviation scenarios* in the following.

In the following, we provide a more detailed explanation of the creation of TBSTs. First, we show how a deviation tree is generated on the basis of experiences. Afterwards, we outline how a so-called *demand tree* is derived from a given deviation tree and a demand prediction. Finally, we sketch how the structure of TBSTs can be further improved.

Creating a Deviation Tree

The creation of a deviation tree is a four-step process that is reminiscent of the creation of a histogram in which dependencies in a sequence of observed deviations are taken into account. In each step, we refer to Figure 5.1 that illustrates the process of how an intermediary derives a deviation tree from the deviations it observed for its subordinate non-dispatchable agents. Hereinafter, we regard the creation of deviation trees for a single agent. A group of agents can be handled in the same manner.

1. **Gather and Normalize Experiences:** A deviation tree is based on a set of sequential experiences, each of which captures a sequence of deviations between the actual and the predicted demand. Each deviation is normalized by means of the normalization function \mathfrak{N} we introduced in Equation (4.1b) to an interval \mathfrak{D} . Here we use $\mathfrak{D} = [-1, 1]$ to capture positive as well as negative deviations. Regarding the first atomic experience in Experience 1 (cf. step 1 in Figure 5.1), for instance, the deviation of $35 - 85 = -50$ between the actual and the predicted demand is normalized

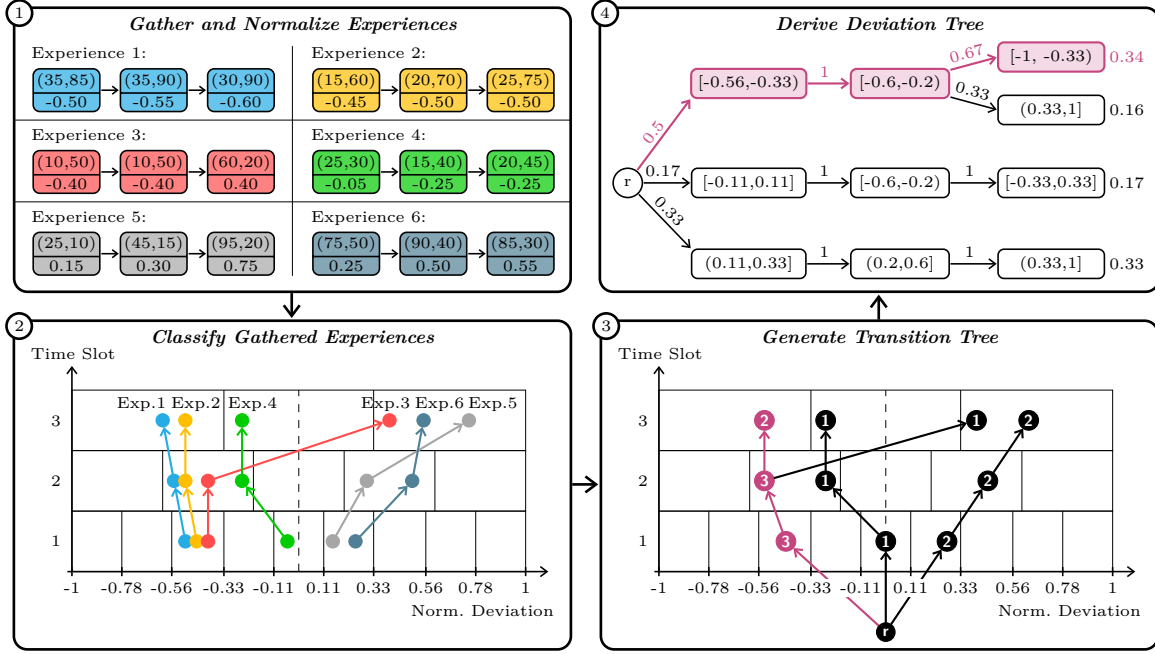


Figure 5.1: The creation of a *deviation tree* is a four-step process: In step 1, a number of sequential experiences is normalized (here we use an interval $[-1, 1]$ and a normalization factor of 100). Afterwards, the normalized sequential experiences are classified by means of a grid of bins while maintaining the information about sequential dependencies. A set of classified experiences that results in the same sequence of bins constitutes a *deviation scenario*, as is the case with Experiences 1 and 2 (cf. step 2). In step 3, we regard the set of deviation scenarios as a tree whose branches result from common prefixes of the sequences of bins (each path from the root r to a leaf represents a deviation scenario). The numbers in the tree's nodes indicate how often a bin is reached on a specific path starting at the root. By using relative instead of absolute frequencies, we obtain, for each inner node, a relative frequency distribution of its child. The result is a deviation tree with conditional probabilities as shown at the tree's transitions in step 4. A scenario's probability of occurrence is equal to that of the corresponding leaf. For example, the probability of the uppermost deviation scenario $\langle [-0.56, -0.33], [-0.6, -0.2], [-1, -0.33] \rangle$ in step 4 is ≈ 0.34 .

to $\mathfrak{N}(-50) = \frac{-50}{100} = -0.5$ (using a normalization factor of 100). That way, we transform each gathered sequential experience E to a *normalized sequential experience* \tilde{E} . With regard to our example, Experience 1 is normalized to $\tilde{E} = \langle -0.50, -0.55, -0.60 \rangle$. To comply with variable agent behavior, we can again apply the concept of a sliding window so that a predefined number of the newest sequential experiences is used to create the deviation tree (cf. our basic trust model in Section 4.2).

- Classify Experiences:** Each normalized sequential experience \tilde{E} is classified by means of a grid of bins (i.e., intervals) of predefined size (cf. step 2 in Figure 5.1). Classifying *sequences* of deviations allows an intermediary to record dependencies in a sequence of observed deviations.

For convenience, we define that each entry of a demand prediction or sequential experience is associated with a specific *time slot* $s \in \mathcal{S} = \{1, \dots, N\}$. We write $[\tilde{E}]_s$ to refer to the s -th entry in normalized sequential experience \tilde{E} . With regard to the grid, each time slot corresponds to a certain row. For each time slot $s \in \mathcal{S}$, we partition \mathfrak{D} in a set of consecutive and disjoint bins $B_s \in \mathcal{B}_s$ such that each possible normalized deviation $\tilde{d} \in \mathfrak{D}$ can be assigned to exactly one bin in \mathcal{B}_s (i.e., $\forall \tilde{d} \in \mathfrak{D} : \forall s \in \mathcal{S} : \exists! B_s \in \mathcal{B}_s : \tilde{d} \in B_s$). In our example, we split \mathfrak{D} into nine intervals of width $\frac{2}{9} \approx 0.22$ for $s = 1$. Hence, we have $\mathcal{B}_1 = \{[-1.00, -0.78], [-0.78, -0.56], [-0.56, -0.33], [-0.33, -0.11],$

$[-0.11, 0.11], (0.11, 0.33], (0.33, 0.56], (0.56, 0.78], (0.78, 1.00]\}$. Suitable sizes of these bins depend on the application. Ultimately, the grid's granularity not only influences the number of TBSs, the shape of the TBST, and the conditional probabilities, but also the quality of derived expectations. If the quality of long-term predictions is less important than those of short-term predictions, one can decrease the bins' granularity with the temporal distance as shown in our example. There, we have nine, five, and three bins for the first, second, and third time slot, respectively.

Each *classified sequential experience* $\hat{E} = \langle B_1, \dots, B_N \rangle$ is a sequence of N bins $B_s \in \mathcal{B}_s$ (with $s \in \mathcal{S}$) that contain the corresponding normalized atomic experience $[\hat{E}]_s$ of the sequential experience E . In our example, we have $\hat{E} = \langle [-0.56, -0.33], [-0.6, -0.2], [-1, -0.33] \rangle$ for Experience 1.

A set of classified sequential experiences that result in the same sequence of bins constitutes a *deviation scenario* Ξ , as is the case with Experiences 1 and 2 in Figure 5.1. The probability of occurrence $p(\Xi)$ of a deviation scenario Ξ is given by the relative frequency of classified sequential experiences \hat{E} mapping to Ξ . With regard to Figure 5.1, the deviation scenario $\Xi = \langle [-0.56, -0.33], [-0.6, -0.2], [-1, -0.33] \rangle$ has a probability of occurrence of $p(\Xi) = \frac{2}{6} = \frac{1}{3} \approx 0.34$.

3. Generate Transition Tree:

We obtain a so-called *transition tree* by regarding the set of deviation scenarios as a tree whose branches result from deviation scenarios with common prefixes (considering the deviation scenarios as sequences of bins). Because we assume that all sequential experiences have the same length N , all leaves of the tree share the same depth N .

Within this tree, each path from the root r to a leaf represents a deviation scenario (note that the root is added to obtain a tree but not contained in any deviation scenario). Consequently, there is a node in a bin B_s and a path $\langle B_1, \dots, B_s \rangle$ (with $s \in \mathcal{S}$) if and only if there is a deviation scenario Ξ whose prefix $[\Xi]^s = \langle B'_1, \dots, B'_s \rangle$ of length s is identical to the path $\langle B_1, \dots, B_s \rangle$ (here, $[X]^j$ denotes the prefix of length j of a sequence X). Note that there might be multiple nodes per bin. Each node is therefore identified by the path by which it is reached. A node that is identified by the path $\langle B_1, \dots, B_s \rangle$ states that the corresponding agent showed a behavior classified as B_s after having shown behaviors classified as $\langle B_1, \dots, B_{s-1} \rangle$ in that order. In step 3 in Figure 5.1, for example, there are two nodes in the bin $[-0.6, -0.2]$ in time slot 2: While the light node is identified by the path $\langle [-0.56, -0.33], [-0.6, -0.2] \rangle$, the black node is identified by the path $\langle [-0.11, 0.11], [-0.6, -0.2] \rangle$.

The transition tree contains information about how often a bin is reached on a specific path that starts at a bin in the first time slot (cf. the numbers in the tree's nodes in step 3 in Figure 5.1). Given a node in a bin B_s that is reached by the path $\langle B_1, \dots, B_s \rangle$ and the set of classified sequential experiences $\hat{\mathcal{E}}$, we assign the number $k = |\{\hat{E} \mid \hat{E} \in \hat{\mathcal{E}} \wedge [\hat{E}]^s = \langle B_1, \dots, B_s \rangle\}|$, which is equal to the number of classified sequential experiences $\hat{E} \in \hat{\mathcal{E}}$ that share the prefix $\langle B_1, \dots, B_s \rangle$.

4. **Derive Deviation Tree:** For each inner node, the relative frequencies of reaching the bins of the next time slot define a node-specific relative frequency distribution of its child. The result is a deviation tree, i.e., a probabilistic model of an agent's behavior, featuring conditional probabilities that a deviation changes from one value (bin) to another. In step 4 in Figure 5.1, the conditional probabilities are shown at the TBST's transitions. With $p(r) = 1$ the probability of the TBST's root r , we define the probability $p(n)$ that a node $n \neq r$ occurs as $p(n) = p(n \mid f(n)) \cdot p(f(n))$, which is the product over all conditional probabilities $p(n' \mid f(n'))$ assigned to the edges of the path identifying n (the function f returns the parent of a node n). Hence, a TBS's probability of occurrence is equal to that of the corresponding leaf. In step 4 in Figure 5.1, the probability of the uppermost deviation scenario $\langle [-0.56, -0.33], [-0.6, -0.2], [-1, -0.33] \rangle$ is $0.34 \approx 0.50 \cdot 1.00 \cdot 0.67$.

Deriving a Demand Tree from a Deviation Tree

When using a deviation tree as predictive model, each TBS embodies a *corridor of expected behavior*. That is because each TBS is a sequence of bins (i.e., intervals) and each bin $B_s = [B_s^l, B_s^u]$ defines a lower B_s^l and an upper bound B_s^u used for classifying normalized experiences. By taking the midpoint

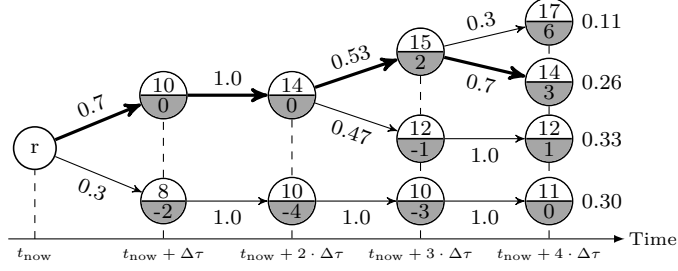


Figure 5.2: An illustration of a deviation tree (lower part of the nodes) and a demand tree (upper part of the nodes) of an intermediary λ obtained from a demand prediction made for the scheduling window $\mathcal{W} = \{t_{\text{now}} + \Delta\tau, \dots, t_{\text{now}} + 4 \cdot \Delta\tau\}$. Each path from the root r to a leaf is a TBS. As for the demand tree, every node (except for the root r) represents an expected demand in the corresponding future time step. The expected demand is calculated by adding a TBS’s expected deviations (lower part of the node) to the predicted demand $\langle 10, 14, 13, 11 \rangle$. The values at the TBST’s edges indicate the conditional probabilities that the demand changes from one value to another. A TBS’s probability of occurrence is equal to that of the corresponding leaf. The highlighted path indicates the *anticipated scenario*, which denotes the most likely development of the demand.

$\frac{B_s^l + B_s^u}{2}$ of each bin B_s contained in a TBS and using the inverse \mathfrak{N}^{-1} of the normalization function for denormalization, we obtain an expected sequence of deviations. For example, the uppermost TBS $\langle [-0.56, -0.33], [-0.6, -0.2], [-1, -0.33] \rangle$ in step 4 in Figure 5.1 represents an expected sequence of deviations of $\langle -44.5, -40.0, -66.5 \rangle$ with probability 0.34 (the bins’ midpoint were multiplied by the normalization factor of 100). The expected deviation -44.5 results from $\frac{-0.56 - 0.33}{2} \cdot 100$, for instance. Obviously, the smaller the size of an interval B_s , the narrower the corridor and the higher the theoretical accuracy of an expected deviation.

With regard to the resource allocation problem, each intermediary uses its deviation tree – reflecting the uncertainties in its local demand – to derive a so-called *demand tree* consisting of *demand scenarios*. In the demand tree, each scenario represents an expected development of the demand. An intermediary obtains a demand tree by adding the predicted demand to each expected sequence of deviations contained in the deviation tree. This process is similar to the calculation of the expected demand based on a demand prediction and a trust value as described in Section 4.2. For instance, adding the deviation scenario $\langle -44.5, -40.0, -66.5 \rangle$ calculated above to the demand prediction $\langle 85, 90, 90 \rangle$ depicted in Figure 4.2b yields the demand scenario $\langle 40.5, 50, 23.5 \rangle$ labeled “expected” in Figure 4.2b.

Figure 5.2 depicts an example of how a demand tree is derived from a deviation tree and a demand prediction. Note that the demand tree has the same shape as the corresponding deviation tree. Within a TBST, the so-called *anticipated scenario* represents the most likely development of an agent’s behavior. With regard to a demand tree, this is the most likely development of the demand. The anticipated scenario results from, starting at the TBST’s root, taking the most probable alternative (i.e., the successor of highest conditional probability) at every node. Hence, the anticipated scenario shows the lexicographically largest sequence of conditional probabilities. As shown in Figure 5.2, the anticipated scenario (26 %) is not necessarily the most likely scenario (33 %). A TBST’s shape and the node-specific frequency distributions are indicators of aleatoric uncertainties: The more similar the conditional probabilities of a node’s children are and the more dissimilar the bins these children represent, the higher is the uncertainty of how the behavior develops.

In Part IV, we borrow and adapt techniques from the field of *online stochastic optimization* [83] to create TBST-based schedules that enable the system to deal with different possible developments of the demand. Together with the ability to choose the most suitable scenario at runtime, such schedules avoid situations in which deviations between supply and demand are higher than (technically) allowed, or where their compensation is either very costly or not feasible due to the dispatchable agents’ inert behavior. Consequently, TBSTs serve as a basis for obtaining *robust solutions* [42] in the sense that the system can efficiently and effectively achieve its goals despite disturbances.

Improving the Structure of Trust-Based Scenario Trees

Depending on the situation and application in which TBSTs are applied, there are several ways to further improve a TBST's structure. We briefly present some of these possibilities in the following. Most of them are used in the evaluation in Section 5.3.

Recent Behavior. Given that TBSTs are especially suited for situations in which an agent's behavior depends on its behavior in previous time steps, the quality of the provided expectations can be further improved by selecting relevant sequential experiences for the generation of a TBST on the basis of its behavior in recent time steps. For example, if the accuracy of the residual load prediction depends on the prediction error of the last k time steps, then it would be beneficial to take the prediction error measured for these time steps into account when creating the TBST.

This is achieved by adding the agent's *recent deviations* to the front of each demand prediction $\hat{D}_{t_{\text{now}}}^w$. The recent deviations are the newest atomic experiences $E[t]$ gained for the last k time steps $t \in \{t_{\text{now}} - (k-1) \cdot \Delta\tau, \dots, t_{\text{now}}\}$. That way, each sequential experience is extended by a "look back" assessing the agent's prediction quality right before the corresponding prediction was made. Each time a new deviation tree is generated, we first generate a TBST that includes the look back. Then, we use the current recent deviations to identify and extract the most appropriate subtree that does not contain the look back. The most appropriate subtree consists of those TBSs whose look back is, regarded in reverse order, lexicographically closest to the recent deviations. We define the distance of a bin $B_s = [B_s^l, B_s^u]$ representing an element of classified look-back behavior and a recent deviation x as the absolute difference between the interval's midpoint and x (i.e., $\left| \frac{B_s^l + B_s^u}{2} - x \right|$). In terms of HMMs, this procedure corresponds to identifying the most likely sequence of hidden states (i.e., the closest TBSs) based on a given sequence of observations (i.e., the recent behavior) in order to determine the HMM's current state. We make use of this feature in our evaluation in Section 5.3.

Time of Day as Trust Context. There are situations in which an agent's behavior depends on the time of day, as is the case with the accuracy of a solar power plant's output predictions. To comply with this dimension of time-dependent behavior, the creation of TBSTs can be extended by a trust context that captures the time frame in which a sequential experience was gained. When creating a TBST, this context is used to identify the relevant experiences.

Limited Number of Scenarios. If a TBST serves as input for an optimization problem whose complexity depends on the number of TBSs (e.g., the scheduling problem), the number of TBSs contained in the TBST can be limited to manage the optimization problem's complexity. In such a situation, one can iteratively remove the least important scenario until the desired number of TBSs is reached. The least important scenario shows the lexicographically smallest sequence of conditional probabilities. After an appropriate number of TBSs has been removed, the conditional probabilities have to be recalculated. Apart from such a greedy approach, one could also use the scenario reduction techniques discussed in [78] to reduce the size of the TBST.

Assumption of Full Compliance in Case of High Uncertainty. If the conditional probabilities of each set of siblings in the TBST are very similar, there is only very little knowledge about how an agent might behave in the future. The *Shannon entropy* serves as a suitable measure to identify such situations of high (aleatoric) uncertainty. In the worst case – which corresponds to the highest possible entropy –, the nodes' frequency distributions of their children are discrete uniform distributions. In such situations, one might be interested in adding a TBS to the TBST that represents the case in which the agent is expected to comply with its predicted behavior (i.e., assuming benevolent behavior).

Removal of Infrequent Scenarios. TBSs that occurred very infrequently in comparison to other scenarios might be regarded as outliers. Such TBSs can be filtered out in the course of the creation of a TBST.

5.3 Evaluation in the Decentralized Power Management Case Study

We split our evaluation into two parts: In the first part, we compare the quality of expectations obtained by TBSTs to the quality of those derived from the basic trust model introduced in Section 4.2. The

purpose of the first part is to demonstrate the advantage of TBSTs over a single trust value in the context of anticipating uncertainties in demand predictions made for multiple time steps. The second part compares TBSTs and HMMs in terms of their ability to provide accurate expected deviations. As discussed in Section 5.1, HMMs are especially suited for recognizing patterns and making forecasts by approximating the stochastic model of a system whose states can only be partially observed. Moreover, we demonstrate the advantage of being able to expect different deviation scenarios instead of relying on a single expectation. All evaluations are performed in the context of the power management case study.

Comparison of Trust-Based Scenarios to Trust Values

In this part of the evaluation, we regard a setting consisting of a single AVPP and an agent, called *predictor*, that represents all consumers as well as the AVPP’s subordinate non-dispatchable power plants (solar plants and wind generators). We use real world data for the capabilities of physical power plants¹ (such as production boundaries), the load curves², and the simulated weather conditions³ influencing the output of weather-dependent power plants. The evaluation is implemented in a sequential, round-based execution model in which each round corresponds to a specific time step, each representing a time span of 15 min. In each time step, the predictor provides the AVPP with a residual load prediction for the next $N = 32$ time steps (i.e., we have 32 time slots per prediction). For each time slot, the residual load prediction was created by adding a randomly generated prediction error to the actual residual load. Each prediction error was generated using a Gaussian distribution. To reflect sequential dependencies in the predictions’ accuracy (as is the case with the quality of weather predictions, for instance), each prediction error depended on previous prediction errors. The stochastic process was further designed in a way that short-term predictions are more precise than long-term predictions, and that predictions became more accurate as a future point in time approaches.

The AVPP’s objective was to minimize the deviation between the actual and the expected residual load. To determine the expected development of the residual load, the AVPP employed the concept of TBSTs and, for comparison, the basic trust model introduced in Section 4.2 that uses a single trust value to capture the mean prediction error. The quality of the predicted residual load serves as baseline. We opted for the basic trust model that uses a single trust value in this evaluation in order to demonstrate the necessity of taking the prediction horizon as trust context into account. As for TBSTs, the prediction horizon is an integral part of the trust model. The TBSs and the trust value were determined on the basis of the 50 newest sequential experiences so that the models could quickly adapt to changes in the quality of predictions. We identified this parametrization in a preceding parameter search. Both TBSs as well as the basic trust model were completely trained at runtime. To capture positive and negative deviations, the AVPP normalized experiences by means of Equation (4.1b) to the interval $[-1, 1]$. The AVPP used the prediction error of the last two time steps as look back to identify relevant TBSs. Further, the AVPP chose the anticipated TBS as expectation of how the prediction error develops. To evaluate the influence of the number of bins on the quality of derived expectations, we ran experiments with $|\mathcal{B}_s| \in \{45, 91, 183, 365\}$ for all time slots $s \in \{1, \dots, 32\}$. For each parametrization, we performed 100 simulation runs over 1000 time steps each, corresponding to a simulated time frame of approximately 10.5 days.

With regard to Table 5.1, we first notice that both TBSTs as well as the basic trust model are able to substantially reduce the deviation between the expected and the actual residual load in comparison to the situation in which the AVPP relied on the residual load prediction. Compared to the unmodified residual load prediction, the mean deviation μ_δ between the expected and the actual residual load can be reduced by approximately 70 % when using trust values compared to approximately 84 % when using TBSs with 183 bins per time slot. This also demonstrates that TBSTs are able to significantly improve the quality of expectations in comparison to a trust value. For 183 bins per time slot, TBSs reduce the trust value’s μ_δ by 46 %. Moreover, TBSs considerably lower the standard deviation σ_δ of

¹Energymap (Bavaria), 2012: <http://www.energymap.info>, retrieved in 2012.

²LEW, 2012: <http://www.lew-verteilnetz.de>, retrieved in 2012.

³LfL (Bavaria), 2010: <http://www.lfl.bayern.de/agm/>, retrieved in 2012.

Method	Number of Bins per Time Slot	Size of Bins [kW]	μ_δ [kW]	σ_δ [kW]	\min_δ [kW]	\max_δ [kW]
Unmodified Prediction	–	–	629.1	32.8	573.6	707.3
Trust Value	–	–	188.9	34.9	121.2	273.2
Trust-Based Scenarios	365	50	100.4	14.6	69.0	160.6
	183	100	101.7	14.0	73.0	152.5
	91	≈ 201	109.0	14.7	76.5	165.6
	45	≈ 405	130.2	17.5	91.6	185.0

Table 5.1: Deviations between the actual and the expected residual load obtained with different methods for determining the expectation: The expected residual load is either (1) equivalent to the predictor’s unmodified residual load prediction, (2) based on a trust value, or (3) derived from TBSs. Here, \min_δ denotes the minimum deviation between actual and expected residual load, \max_δ the maximum, μ_δ the average, and σ_δ the corresponding standard deviation.

the expectations’ quality from 34.9kW to 14.0kW. Note that TBSs compensate for about 50 % of the standard deviation of the residual load prediction, thereby lowering the expectation’s uncertainty.

These observations can also be made with regard to Figure 5.3a, which depicts the deviation of the expected from the actual residual load over time. TBSs not only achieve the lowest deviations but also the most stable quality of expectations. Figure 5.3b illustrates the advantage of TBSs over the basic trust model when trying to quantify and anticipate prediction errors that increase with the temporal distance to the time slot a prediction is made for. The increase of the prediction error with the prediction horizon is reflected in the curve “Unmodified Prediction”. Using a single trust value reflecting the mean prediction error results in a predictive model that can provide accurate expectations for a specific prediction horizon (here approximately between time slot 11 and 21). On average, expectations provided for smaller or greater time slots are significantly worse. Because each TBS describes the predictor’s behavior as a sequence of expected deviations, the predictor’s time-slot-dependent prediction quality can be perceived and anticipated in much greater detail, resulting in a much more stable quality of expectations.

Besides the reduction of the mean deviation, it is also very important to reduce the maximum deviation \max_δ between the expected and actual residual load. Trust values reduce \max_δ by 61 %. TBSs obtain 78 % so that their \max_δ is 44 % lower than the trust value’s \max_δ .

Last but not least, we expected that TBSs benefit from a large number of bins per time slot. While Table 5.1 confirms our expectation (cf. the mean deviation μ_δ for the different number of bins), it also shows that 183 bins were sufficient to model the underlying stochastic process.

Summarizing, compared to our basic trust model, TBSs significantly increased the AVPP’s ability to anticipate the residual load’s prediction error. Moreover, the risk the AVPP is exposed to, measured as maximum deviations, decreased considerably and the variation in the expectations’ quality declines. Consequently, TBSTs not only obtain a higher average quality of expectations but also a higher confidence (in the sense that the trust model describes the actual observable behavior of the residual load) than the employed basic trust model.

Comparison of Trust-Based Scenarios to Hidden Markov Models

In the second part of our evaluation, we compare TBSTs to HMMs with regard to the accuracy of the forecasted expected deviations. For this purpose, we used a Java-based implementation of first-order HMMs called *Jahmm*⁴. *Jahmm* implements the forward-backward algorithm to calculate the probability that the HMM generates a given sequence of observations, the Viterbi algorithm to identify the most likely sequence of hidden states based on a given sequence of observations, and the Baum-Welch algorithm to determine the HMM parameters. These parameters comprise the number of hidden states, the means and the standard deviations of the states’ Gaussian distributions, as well as the state transition probabilities.

⁴Jahmm, Version 0.6.2, 2011: <https://code.google.com/archive/p/jahmm/>, retrieved on March 1, 2016.

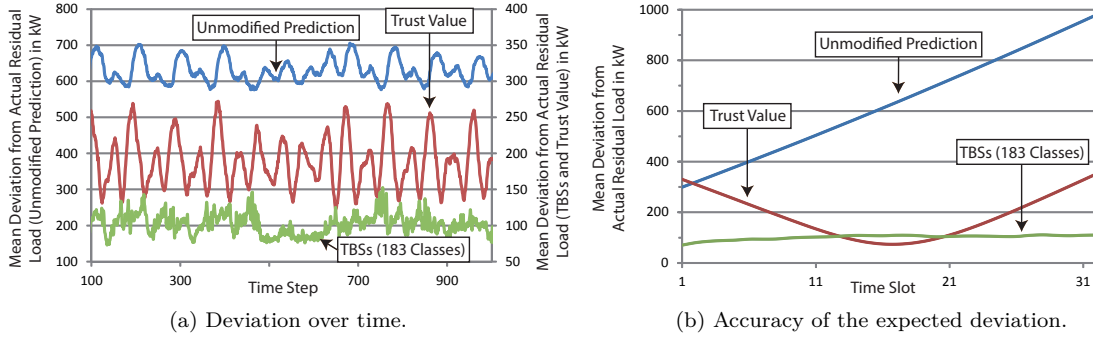


Figure 5.3: Mean deviation of the expected residual load (unmodified residual load prediction, trust value, TBSs) from the actual residual load. Figure 5.3a depicts this deviation over time. Figure 5.3b shows the accuracy of the expected residual load for the 32 time slots. The curve “Unmodified Prediction” in Figure 5.3b demonstrates that the prediction error increased with the prediction horizon, i.e., the time slot a prediction was made for.

In this evaluation, we considered a similar evaluation setting as in the first part of our evaluation. This time, we regard five AVPPs instead of a single AVPP, and the accuracy of expected deviations for the next $N = 4$ time steps, though. The composition of each AVPP was generated at random, based on a set of 173 dispatchable and 350 weather-dependent power plants. We ran a single experiment over five simulated days, which corresponds to 480 time steps using a resolution of 15 min per time step. For each day, the prediction error varied with the time of day (forenoon, afternoon, night).

In a preceding parameter search, we optimized the HMMs’ maximum number of hidden states for this specific evaluation setting. A maximum number of 14 states turned out to be beneficial because it allows the HMM to form a precise model of typical residual load deviations on the one hand, but avoids overfitting on the other hand. As for TBSTs, we made use of some of the different possibilities for improving a TBST’s structure (see Section 5.2): We considered the last three deviations as look-back behavior for the selection of the most appropriate subtree, limited the TBSTs’ size to 20 scenarios, implemented a trust context for the time of day to identify the up to 40 newest sequential experiences that were gained at a similar time of day, and added a TBS that assumes benevolent behavior to a TBST if the conditional probabilities for the root’s children were identical. With regard to the grid of bins, we used 250 bins for the first time slot, 50 bins for the fourth time slot, and a linearly decreasing number of bins for the slots between them. Thereby, we considered long-term prediction quality less important than short-term prediction quality. As for the classification of look-back behavior, we used 150 bins for each time slot.

The first four of the five simulated days were dedicated for gathering training data for the TBSTs and the HMMs. The quality of expected deviations is evaluated in the course of the fifth day. But also during the fifth day, further training data is gathered. In each time step $t \in \{1, \dots, 480\}$, each AVPP’s HMM was trained on the basis of the full sequence of the t observed prediction errors. We used the same sequence to determine a HMM’s current state by means of the Viterbi algorithm. HMMs forecasted a sequence of expected deviations for the next $N = 4$ time steps as follows: Starting at the current state, a HMM recursively chose the most likely transition at a visited state. The sequence of expected deviations results from the sequence of visited states. The mean value of a visited hidden state’s Gaussian distribution serves as expected deviation. Note that the procedure of taking the most probable alternative at every state corresponds to the definition of a TBST’s anticipated scenario.

Table 5.2 shows that both TBSTs as well as HMMs are able to anticipate the weather-dependent power plants’ behavior and thus to significantly reduce the prediction error of the residual load. As for HMMs, the relative reduction compared to the residual load prediction ranges between $\mu_{\%} = 77.0\%$ (standard deviation $\sigma_{\%} = 38.6\%$) for the first time slot and $\mu_{\%} = 71.6\%$ ($\sigma_{\%} = 39.2\%$) for the fourth

Time Slot	Relative Reduction	HMMs	Anticipated TBS	TBSTs	
				Most Probable TBS	Best Node
1	$\mu_{\%}$ [%]	77.0 (1.5)	77.2 (0.7)	65.8 (2.5)	93.0 (0.6)
	$\sigma_{\%}$ [%]	38.6 (3.0)	39.4 (2.7)	46.8 (2.1)	5.6 (1.0)
2	$\mu_{\%}$ [%]	72.2 (7.7)	73.4 (0.8)	62.6 (3.0)	88.8 (0.7)
	$\sigma_{\%}$ [%]	59.4 (40.4)	39.6 (2.2)	49.0 (4.6)	15.8 (1.0)
3	$\mu_{\%}$ [%]	70.2 (1.2)	64.8 (2.2)	58.2 (3.5)	85.2 (0.7)
	$\sigma_{\%}$ [%]	40.0 (3.1)	42.8 (2.7)	49.6 (3.6)	19.8 (0.4)
4	$\mu_{\%}$ [%]	71.6 (1.0)	71.8 (1.9)	66.4 (2.4)	80.2 (1.0)
	$\sigma_{\%}$ [%]	39.2 (3.0)	39.8 (2.2)	43.2 (1.2)	26.8 (4.6)

Table 5.2: Relative reduction of the deviation between the expected and the actual residual load, compared to the residual load prediction for $N = 4$ time slots. The data is based on expectations and predictions the five AVPPs provided at daytime, which comprises 60 of the 96 time steps of the fifth simulated day (we deliberately omit the results obtained at night because these data do not allow us to derive reasonable values for the relative reduction for technical reasons). All listed values are based on 300 data points. For each AVPP, we calculated the average relative reduction of the prediction error over the 60 time steps at daytime, as well as the corresponding standard deviation. Here, $\mu_{\%}$ denotes the mean of the five average values (one per AVPP), and $\sigma_{\%}$ the average of the five corresponding standard deviations. Values in parentheses denote standard deviation.

time slot on average. Regarding the anticipated scenario, TBSTs obtain similar values ranging between $\mu_{\%} = 77.2\%$ ($\sigma_{\%} = 39.4\%$) for the first time slot and $\mu_{\%} = 71.8\%$ ($\sigma_{\%} = 39.8\%$) for the fourth time slot. The observation that the relative reduction of the prediction error tends to decline with the time slot, i.e., the prediction horizon, complies with our expectation. In case of TBSTs, this can also be attributed to the fact that we used less bins to classify experiences in greater time slots. These results show that the TBSTs' anticipated scenario yields a similar quality of expectations as HMMs.

While HMMs obtain a better mean relative reduction $\mu_{\%}$ than TBSTs for the third time slot (70.2% compared to 64.8%), TBSTs achieve a much more stable quality of expectations $\sigma_{\%}$ for the second time slot (39.6% compared to 59.4%). The latter can be ascribed to the TBSTs' ability to differentiate between multiple scenarios.

The advantage of distinguishing between multiple scenarios is also reflected in Figure 5.4 which shows the development of the absolute deviation between the expected and the actual residual load over the fifth day. When the weather-dependent power plants' prediction error changes with the sunset, HMMs have much more difficulties in delivering an adequate expected deviation than TBSTs. This is reflected in the spike at time step 38.

Table 5.2 further confirms our expectation that the anticipated TBS provides a better indication of the future than the most probable TBS (see Section 5.2). The most probable TBS only yields a relative reduction between $\mu_{\%} = 65.8\%$ ($\sigma_{\%} = 46.8\%$) for the first time slot and $\mu_{\%} = 66.4\%$ ($\sigma_{\%} = 43.2\%$) for the fourth time slot on average. The relative reduction is thus between 85% and 92% of the one obtained with anticipated TBS.

A very interesting aspect of maintaining multiple scenarios in the form of a TBST is revealed when we allow the AVPPs to switch to the TBSTs' node whose expected prediction error is closest to the actual prediction error (cf. "Best Node" in Table 5.2 and Figure 5.4). In this case, TBSTs obtain an average relative reduction between $\mu_{\%} = 93.0\%$ ($\sigma_{\%} = 5.6\%$) for the first time slot and $\mu_{\%} = 80.2\%$ ($\sigma_{\%} = 26.8\%$) for the fourth time slot. The relative reduction is between 130% and 112% of the one achieved with the anticipated TBS or the HMMs. In Part IV, we make use of this characteristic by creating schedules on the basis of an entire TBST instead of a single TBS. For the dispatchable agents, these schedules serve as blueprints for how many resources to provide in which situation.

To summarize, the TBSTs' anticipated TBS provides a similar quality of expectations as HMMs. The advantage of considering a whole tree instead of a single scenario becomes evident when regarding complex agent behavior that changes over time, or when enabling the agents to identify and switch to

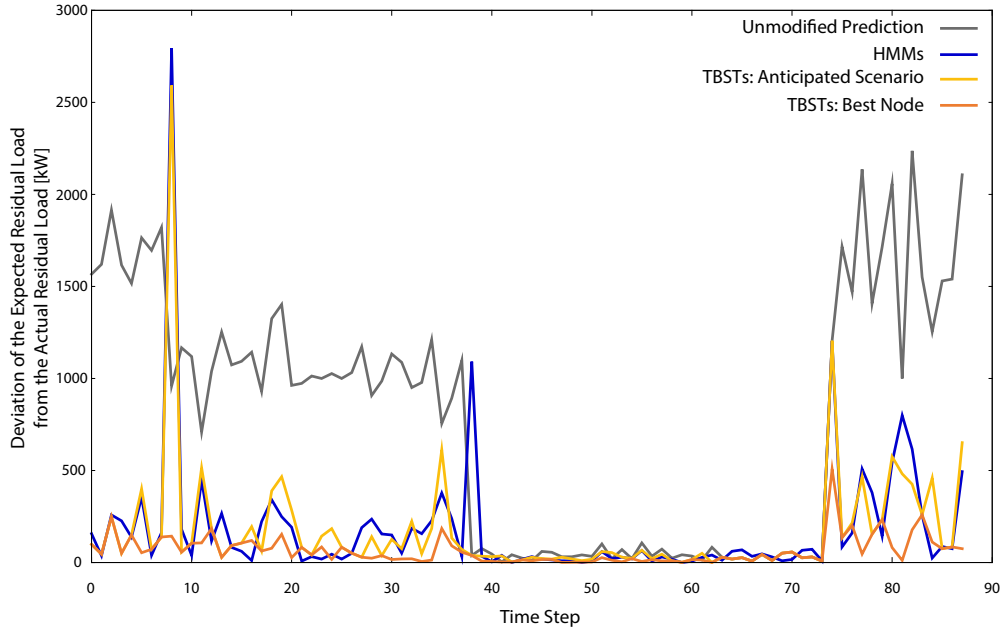


Figure 5.4: Development of the absolute deviation of the expected residual load from the actual residual load for a single AVPP and the first time slot over the fifth simulated day. TBSTs and HMMs substantially reduce the absolute deviation. “Best Node” indicates the deviation between expected and actual residual load when allowing the AVPP to switch to the TBSTs’ node that is closest to the actual prediction error at runtime.

the most suitable TBS contained in a TBST at runtime.

Chapter Summary and Outlook

In this chapter, we introduced the concept of *Trust-Based Scenario Trees* (TBSTs) as a means to anticipate agent behavior that follows different behavioral patterns and exhibits sequential dependencies in a series of observations. TBSTs approximate such behavior based on a set of *Trust-Based Scenarios* (TBSs) that specify a discrete probability distribution over a series of time steps. The process of creating a TBST is reminiscent of the creation of a histogram in which dependencies in a sequence of observed behavior are taken into account. Each TBS has a certain probability of occurrence and represents a possible development of an agent’s behavior. Compared to other trust models, this allows for a more fine-grained representation of an agent’s behavior. As for the resource allocation problem regarded in this thesis, TBSTs are used to anticipate uncertainties in demand predictions. Each TBS represents an expected series of deviations from a demand prediction based on previously observed deviations.

Our evaluation demonstrated that TBSTs are better predictive models than the basic trust model we introduced in Section 4.2. Further, a TBST’s anticipated TBS obtains a similar quality of expectations as a hidden Markov model – a well-known approach to approximate the stochastic model of a system whose states can only be partially observed. We revealed the full advantage of TBSTs in a situation in which we allowed the agents to identify and switch to the most suitable TBS contained in a TBST at runtime.

In future work, we want to investigate combinations of our concept of TBSTs with other approaches to predictive modeling. Based on an agent’s recent behavior, hidden Markov models or artificial neural networks could be employed, for instance, to improve the selection of the most appropriate subtree in a TBST that includes look-back behavior. Moreover, we want to investigate the integration with scenario reduction techniques from the literature.

In Part IV, we show how intermediaries create TBST-based schedules that enable the system to deal with different possible developments of the demand. In combination with the ability to choose the most suitable scenario at runtime, TBST-based schedules avoid situations in which deviations between supply and demand are higher than (technically) allowed, or where their compensation is either very costly or not feasible because of the dispatchable agents' inert behavior. Due to the computational complexity of solving the scheduling problem, we outline a regionalized auction-based mechanism for scalable schedule creation in large-scale systems, called *TruCAOS*. TruCAOS not only considers uncertain demand by means of TBSTs but also implements a trust-based risk-avoidance strategy that mitigates uncertainties originating from schedule violations.

Part III

Self-Organizing Hierarchical System Structures as Foundation of Robustness and Scalability

In numerous MAS, the task is to solve an optimization problem whose complexity is subject to the number of participating agents. A well-known representative is the scheduling problem considered in this thesis. To achieve scalability in large-scale systems, a crucial step is to establish and maintain an organizational structure that supports the agents' and the system's objectives [92]. Hierarchical system structures come into play if the regarded optimization problem can be reasonably decomposed in a hierarchical manner. In this part, we describe how such hierarchies can evolve in a self-organizing manner and show that they allow the system to autonomously come to a compromise between solution quality and performance in terms of runtime. Essentially, hierarchies are obtained by recursively partitioning a set of agents into disjoint subsystems represented by intermediaries. These are then arranged in a tree. If the system has to deal with uncertainties originating from participants that cannot or should not be excluded from the system, the characteristics of the created partitionings become a major concern – a single subsystem not being able to cope with its uncertainties endangers the goal of the overall system.

To this end, we first introduce the partitioning problem – which is an NP-hard combinatorial optimization problem – as the basis of autonomous problem decomposition and the formation of efficient and stable system structures in Chapter 6. Afterwards, we present two self-organization algorithms that solve the partitioning problem in an efficient and effective manner in Chapters 7 and 8. The first algorithm, called PSOPP, builds on a population-based metaheuristic, whereas the second, called SPADA, is a decentralized agent-based approach. Finally, we sketch a control loop that regulates the formation of hierarchies by means of these algorithms in Chapter 9.

Self-Organizing Partitions for Autonomous Hierarchical Problem Decomposition

Summary. A hierarchical system structure promises scalability with regard to the number of participating agents. By allowing the agents to form such a structure by themselves and at runtime, the system self-improves its ability to deal with uncertain demand and supply. That is because the agents can maintain a structure complying with the current circumstances under which the resource allocation problem has to be solved. One way to implement such a self-organizing hierarchical structure is to enable the agents to recursively solve the so-called partitioning problem. In fact, this problem is at the heart of the formation of several organizational structures in MAS, such as coalitions or teams. In this chapter, we introduce the partitioning problem on the basis of constraints that guide the formation of the hierarchical structure by means of suitable ranges for the number and the size of partitions. As we will demonstrate in Section 9.2, this is of particular interest because these factors not only influence the time the agents need to calculate the schedules but also their quality in terms of accuracy and costs. The *partitioning constraints* thus serve as a means to steer the process of self-organized problem decomposition. Moreover, we introduce a new organizational paradigm, called *homogeneous partitioning*, that yields a structure of similar partitions (i.e., organizations). That way, each organization is equally able to cope with disturbances originating from a highly dynamic and uncertain environment. Homogeneous partitioning increases the hierarchy's stability and promotes the system's robustness in terms of its ability to hold the balance between supply and demand. In our evaluation, we demonstrate that homogeneous partitionings are far more robust against environmental changes than organizations consisting of homogeneous agents.

Publication. The concepts and findings described in this chapter have been published in Anders et al. [9, 15, 16].

In large-scale open technical systems, the complexity of a centralized solution model prohibits obtaining a solution to the scheduling problem in reasonable time. With regard to the power management case study, for instance, the creation of schedules must not last longer than 15 min. Figure 6.1a depicts a situation where a central authority has to calculate the schedules for all dispatchable agents on its own. One way of dealing with complexity and scalability issues is to decompose the overall scheduling problem into smaller, tractable sub-problems that still offer close to optimal solutions to the overall problem. Because the complexity of the scheduling problem depends on the number of agents for which schedules have to be created, such a decomposition can be obtained by forming an adequate organizational structure in which the overall set of agents is distributed among several subsystems (i.e., organizations), each of which represents a smaller sub-problem. A discussion of different organizational paradigms for MAS can be found in [92].

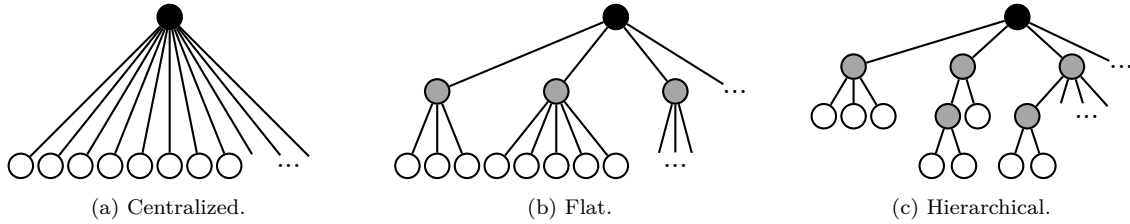


Figure 6.1: In a centralized system, a central authority has control over all other agents in the system and calculates schedules for all dispatchable agents (see Figure 6.1a). By creating subsystems that are represented by intermediaries, we obtain a flat system structure of height 2 that decomposes the scheduling problem into several sub-problems (see Figure 6.1b). The top-level intermediary (cf. the black node) controls all subordinate intermediaries (cf. gray nodes) which, in turn, control the agents representing physical devices (cf. white nodes). A hierarchical system structure allows the system to scale with an arbitrary number of agents by introducing new intermediaries where needed (see Figure 6.1c). Note that the hierarchy’s leaves do not have to have the same depth. In a hierarchy, intermediaries control subordinate intermediaries and agents representing physical devices.

Hierarchical problem decomposition is a well-known method to deal with the complexity of resource allocation in large-scale systems (see, e.g., [2, 40, 69, 115]). Here, the overall scheduling problem is decomposed by forming a *hierarchical structure* of agent organizations [92]. Figure 6.1b illustrates the simplest case of such a hierarchy – a flat structure of height 2 that originates from partitioning the set of agents into disjoint subsystems. Each organization is represented by an *intermediary* (cf. the internal nodes of the hierarchy). The former central authority now acts in the role of the *top-level intermediary*. It is responsible for triggering the schedule creation. As sketched in the context of the power management case study in Section 2.2, the overall scheduling problem is solved in such a structure in a top-down manner. This means that the top-level intermediary calculates schedules for all subordinate intermediaries which, in turn, redistribute the assigned demand to their subordinate dispatchable agents. Each intermediary is thus responsible for solving a part of the overall scheduling problem. Forming organizational structures in which each agent is assigned to a single intermediary leads to a proper problem decomposition and avoids situations in which agents receive conflicting instructions (e.g., schedules) which might cause inconsistent system states.

A flat structure does not scale with an arbitrary number of agents, though. Adding further agents to the system would require to increase the size of the organizations, their number, or both. Note that each of these cases would increase the complexity of solving the scheduling problem for at least one intermediary. A hierarchical structure as shown in Figure 6.1c does not suffer from these scalability issues. To rise the degree of problem decomposition, it is sufficient to introduce new organizations within an existing organization that, again, are represented by intermediaries. Note that the leaves of the resulting hierarchy do not have to have the same depth. In a hierarchical system, intermediaries have to create schedules for all subordinate dispatchable agents, including subordinate intermediaries and dispatchable agents representing physical devices. Such a hierarchical structure in which each intermediary represents an entire subsystem can be regarded as a *system of systems* [182]. In terms of our power management case study, intermediaries correspond to AVPPs and the leaves in the hierarchy represent physical power plants.

Compared to the centralized approach, the intermediaries of a hierarchy have to create schedules for a lower number of dispatchable agents so that the average time an intermediary needs to create schedules decreases. Instances of the scheduling problem that have to be solved by intermediaries residing in different branches of the hierarchy do not depend on each other and can thus be solved concurrently. When determining the scheduling problem’s theoretical (see Section 2.2) or practical computational complexity, the top-down dependency of instances of the scheduling problem requires to sum the intermediaries’ scheduling times for each branch, though. The *maximum sequential scheduling*

time (i.e., the longest of all serial paths that result from adding the scheduling times for each branch in the hierarchy) therefore serves as metric for measuring the time needed to solve the overall scheduling problem (see Equation (6.1)). Given an adequate hierarchical structure, the agents can solve the scheduling problem in less time than the centralized approach. Within the hierarchy, intermediaries can create schedules by means of the regio-central approach sketched in Section 2.2 or the auction-based mechanism explained in Chapter 10, among others. Note that we seek *adequate* hierarchical structures because we do not necessarily acquire shorter maximum sequential scheduling times by increasing the number of subsystems or the height of the hierarchy (cf. our evaluation results in Section 9.2).

Definition 1 (Maximum Sequential Scheduling Time). *The **maximum sequential scheduling time** $MSST$ is defined as the longest of all serial paths that result from adding the scheduling times for each branch in the hierarchy:*

$$MSST = SST(\Lambda) \tag{6.1}$$

$$SST(\lambda) = \begin{cases} ST_\lambda + \max \{SST(\lambda') \mid \lambda' \in \mathcal{I}_\lambda\} & \text{if } \mathcal{I}_\lambda \neq \emptyset \\ ST_\lambda & \text{else} \end{cases}$$

*The maximum sequential scheduling time corresponds to the **sequential scheduling time** $SST(\Lambda)$ of the top-level intermediary Λ , which is calculated by adding the **scheduling time** ST_Λ of the top-level intermediary Λ to the maximum of the sequential scheduling times $SST(\lambda)$ of its subordinate intermediaries $\lambda \in \mathcal{I}_\Lambda$. The sequential scheduling time of an subordinate intermediary is defined recursively. If an intermediary $\lambda' \in \mathcal{I}$ does not control any subordinate intermediaries (i.e., if $\mathcal{I}_{\lambda'} = \emptyset$), its sequential scheduling time $SST(\lambda')$ corresponds to its scheduling time $ST_{\lambda'}$. The maximum sequential scheduling time is thus the lower bound on the runtime achievable by perfect parallelization.*

To be able to assign schedules to intermediaries that can be fulfilled by the represented collective, we need to address the task of describing an intermediary's *composite* control model. We obtain a precise model by taking the Cartesian product of the control models of an intermediary's subordinates. If this direct product was employed for solving the scheduling problem at higher hierarchy levels, the top-level intermediary would have to solve an optimization problem that is based on a fully centralized model. For the sake of scalability, we therefore choose to sacrifice exact overall optimality by using *abstracted models* of collectives that yield tractable sub-problems and allow for close-to-optimal solutions. That means that each intermediary λ provides its superordinate intermediary with a single abstracted model for the entire collective of dispatchable agents \mathcal{V}_λ it represents. Within the hierarchy, intermediaries make coarse decisions about schedules on the basis of these abstracted models. These decisions are refined when the assigned demand is redistributed at lower levels. The box before the next section gives a brief overview of the algorithms we presented to calculate these abstractions.

The characteristics of open systems presented in Chapter 1 (including variable agent behavior and the highly dynamic environment) prohibit the use of predefined system structures that will be outdated sooner or later. Instead, the agents themselves have to find and maintain an adequate hierarchical compartmentalization of the overall system at runtime. This is of special interest because the overall system can only achieve its goals on the macro level if each subsystem is sufficiently able to fulfill its goals on the meso level. In this thesis, we regard a situation in which all subsystems share a common goal, namely satisfying the assigned fraction of the demand. With regard to the power management case study, the satisfaction of the overall load is endangered if a single AVPP cannot provide an output as stipulated in its schedule.

If the hierarchy is created in a self-organizing manner and in compliance with the current circumstances under which the scheduling problem has to be solved, the system is able to find a suitable trade-off between the time needed to create schedules and the solution quality itself. That way, self-organization serves as a dynamic and autonomous form of problem decomposition that scales with the number of agents schedules have to be created for. It ensures that the system's structure stays in tune with the current environmental conditions and the system's internal state. While we focus on the scheduling

problem in this thesis, this approach is not limited to this specific problem. In principle, it can be applied to an entire class of hierarchically decomposable optimization problems whose scalability hinges on the number of agents involved.

In this chapter, we introduce the underlying combinatorial optimization problem the agents have to solve to form scalable and stable hierarchical system structures that allow them to deal with uncertain demand and supply. In Section 6.1, we outline the so-called *partitioning problem* (PP) which constitutes the central problem that has to be solved to obtain self-organizing hierarchical system structures. With regard to our case study, each AVPP stands for a partition and a set of AVPPs that share the same parent represents a partitioning. While the terms “organization”, “subsystem”, and “partition” are more or less interchangeable, we use them depending on the regarded context. To allow the agents to control the shape of the hierarchy and thus the degree of problem decomposition, we extend the PP by *partitioning constraints* that specify ranges for the size and the number of created partitions in Section 6.2. In Section 6.3, we propose a new organizational paradigm, called *homogeneous partitioning*, for the class of open systems considered in this thesis. With regard to the PP, homogeneous partitioning describes the desired composition of the created subsystems for selected formation criteria. It thus serves as (a constituent of the) objective function. Homogeneous partitioning not only aims at creating stable system structures in terms of a low number of necessary reorganizations, but also at promoting the system’s robustness by favoring the creation of subsystems that are able to comply with the uncertainties present in open systems. In Section 6.4, we briefly explain how intermediaries initiate the reorganization of a partitioning within the hierarchy in case of the violation of the corridor of correct behavior, and describe how they adjust the system structure according to the result of the reorganization. Finally, we discuss related work in the context of the PP, self-organizing hierarchical system structures, and hierarchical problem decomposition in Section 6.5.

Based on the findings of this chapter, we introduce two self-organization algorithms, called PSOPP and SPADA, for solving the PP in Chapters 7 and 8. In contrast to related approaches, these algorithms solve the PP in a general manner, meaning that different types of objectives for diverse formation criteria can be combined. Moreover, they create solutions that abide by the partitioning constraints. In Chapter 9, we present a control loop that regulates the formation of hierarchies on the basis of a partitioning algorithm like PSOPP or SPADA.

Model Abstraction: Dynamic Creation of Compositional Control Models

On higher levels, intermediaries decide on schedules that have to be fulfilled by their subordinate agents. To make optimal decisions, exact models of all subordinates in composition would be required – eventually yielding the same complexity as a centralized solution. The goal of *model abstraction* [186] is to create a composite control model for each intermediary that is comparable in calculation efforts to those of agents representing physical devices. Because of the organizations’ dynamic composition, abstraction has to be performed at runtime. The dynamic creation of compositional control models serves as enabler for the self-organized hierarchical schedule creation studied in this thesis. In the following, we briefly revisit the core concepts of the abstraction algorithms we presented in [187, 189]. An in-depth investigation of these algorithms will be part of a forthcoming thesis.

General Abstraction First, the joint space of feasible supply L_λ for an intermediary $\lambda \in \mathcal{I}$ has to be found on the basis of its subordinate dispatchable agents \mathcal{V}_λ . Because each mode of operation (e.g., on/off) constrains feasible supply to a specific range (e.g., a positive supply between minimal and maximal boundaries if a power plant is on), we have to combine all possible modes. Assume that the set of subordinate agents \mathcal{V}_λ consists of two power plants a_i and a_j that both can be turned off but contribute in the intervals [2 MW, 5 MW] and [8 MW, 15 MW], respectively, if they are on. For the collective $\{a_i, a_j\}$, we get [0 MW, 0 MW] if both a_i and a_j are off, [2 MW, 5 MW] if only a_i is on, [8 MW, 15 MW] if only a_j is on, and [2 MW, 5 MW] + [8 MW, 15 MW] = [10 MW, 20 MW] if both a_i and a_j are on. Abstraction enters the picture in the sense that, at a higher level, it does not matter whether, e.g., a joint supply of 10 MW at time step t is obtained by $S_{a_i}[t] = 2 \text{ MW}$, $S_{a_j}[t] = 8 \text{ MW}$ or $S_{a_i}[t] = 0 \text{ MW}$, $S_{a_j}[t] = 10 \text{ MW}$.

We can therefore merge overlapping intervals, resulting in a normalized, sorted list of non-overlapping intervals, e.g., $L_\lambda = \langle [0 \text{ MW}, 0 \text{ MW}], [2 \text{ MW}, 5 \text{ MW}], [8 \text{ MW}, 20 \text{ MW}] \rangle$. Here, we identified $(0 \text{ MW}, 2 \text{ MW})$ or $(5 \text{ MW}, 8 \text{ MW})$ as “holes”, i.e., contributions that cannot be achieved by $\{a_i, a_j\}$ due to discontinuities. This procedure naturally extends to multiple modes (e.g., start-up, shut-down, etc.). Similarly, we obtain lists of feasible intervals for collectives of $k = |\mathcal{V}_\lambda| > 2$ agents. In the worst case, however, $O(m^k)$ intervals are calculated, where m is the largest number of distinct intervals a single agent has. Our application scenario typically leads to $m = 2$ at lower hierarchy levels since a dispatchable power plant a contributes $[0 \text{ MW}, 0 \text{ MW}]$ if it is off and in $[S_a^{\min}, S_a^{\max}]$ with $0 \text{ MW} < S_a^{\min} \leq S_a^{\max}$ when on. In power systems, some power plants may however be so-called “must-run” plants [230], leading to a singleton list $L_\lambda = \langle [S_a^{\min}, S_a^{\max}] \rangle$. Hence, if all dispatchable agents must run, $m = 1$ and thus this single supply interval of the collective can be found in time linear in k . The situation is ameliorated by applying the binary combine-and-merge operation incrementally.

Temporal Abstraction While general abstraction describes feasible regions of an intermediary, it fails to consider momentary *states* of its children, such as their current supply. These determine the intermediary’s inertia (e.g., if some agents are at peak level or switched off, the intermediary cannot ramp up at its maximal rate of change). *Temporal* abstraction calculates infeasible ranges of an intermediary λ for all time steps $t \in \mathcal{W}$ given the current state, such as its current supply $S_\lambda[t_{\text{now}}]$. For each future time step, we perform a *maximization step* as well as a *minimization step* for each child’s supply. This provides us with the feasible regions of all subordinate agents as depicted in Figure 6.2.

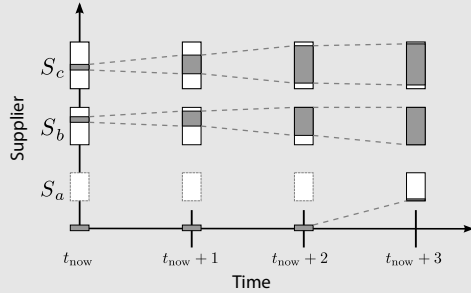


Figure 6.2: Temporal abstraction for an intermediary consisting of three suppliers a, b, c given their current states shown in time step t_{now} . White boxes indicate general boundaries, gray areas represent the temporal boundaries at time step $t \in \{t_{\text{now}}, \dots, t_{\text{now}} + 3\}$. Supplier a needs two time steps to start up and is then available at its minimal output. Adapted from [187].

Similarly to the general abstraction, we merge these regions to find abstract boundaries. With regard to an intermediary λ and a time step t , supply values $S_\lambda[t]$ below or beyond those ranges L_λ^t are guaranteed to be *infeasible*. These intervals *further* constrain feasible schedules in addition to the general boundaries L_λ determined by general abstraction that hold for all time steps and are represented by the combined white intervals in Figure 6.2.

Sampling Abstraction In addition to finding the feasible regions of an intermediary, we are interested in functional relationships between the variables of a collective, such as a cost function mapping joint supply to total (minimal) costs. Similarly, inertia functions for the intermediary depending on its current supply ought to be found. This is necessary because temporal abstraction only excludes *definitely infeasible* contributions for time step $t \in \mathcal{W}$. It does not restrict the transition between two independently feasible but not consecutively reachable contributions for future time steps.

Functional dependencies over the combinatorial domain (e.g., “production to costs” or “production to minimal or maximal next production”) are approximated by repeatedly sampling input-output pairs. With enough points of the function, we aim for an approximation that interpolates sufficiently accurate in unknown ranges of the function such that a simpler representative can be used at a higher level (e.g., a piecewise linear function). However, if the sampled data points are weakly informative, the resulting abstracted high-level optimization introduces severe errors. Furthermore, determining a sampling point comes at the expense of solving an optimization problem as we ask questions like: “What is the *cheapest*

way to schedule power plants a_i and a_j such that their joint supply is 20 MW?” Generally, there are several schedules for subordinates that achieve, e.g., a given joint production at different costs or maximal successor production. In [189], we therefore presented a machine-learning-guided approach to sampling point selection. The algorithm trains a probabilistic regression model with a set of already sampled points. It then proceeds by asking for function values at locations where the regression model is *most uncertain* (see Figure 6.3).

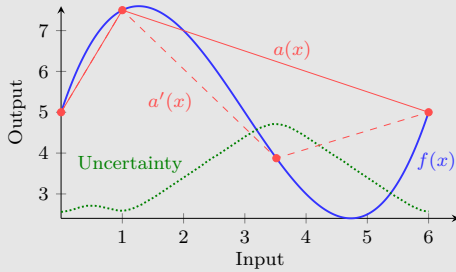


Figure 6.3: Sampling abstraction is concerned with selecting informative points of an unknown but computable function $f(x)$ (bold solid line). Input and output represent variables of a collective, e.g., *production to costs*. An approximation $a(x)$ (solid line) of $f(x)$ is iteratively improved at its most uncertain point to yield a better approximation $a'(x)$ (dashed line). These points are provided by a regression model (dotted line). Adapted from [17].

Possible regressors include Gaussian Processes or Decision Forests. The choice has to be made in accordance with the universe of possible functions a regressor can model (smooth functions, linear functions with jumps etc.). For our considered cost and inertia functions, Decisions Forests with linear leaf models turned out to be more effective than Gaussian Processes due to the presence of discontinuities and jumps [57, 130]. With this improved selection strategy, offering more sampling points indeed led almost monotonically to better results (in contrast to an equidistant strategy) and reduced the overhead costs compared to optimal solutions from 1.7 % to 0.9 % [189].

Summary of Evaluation Results The simulation results presented in [186, 187, 189] confirm that solving the scheduling problem in a hierarchical system and on the basis of abstracted composite control model for each intermediary significantly reduces the time needed to create schedules (see Figure 6.4). The scheduling times of the centralized approach strictly grow faster than the hierarchical ones achieved with the regio-central scheduling approach. The overhead costs for providing the energy incurred by using the hierarchical scheme are in the order of magnitude of 1 %.

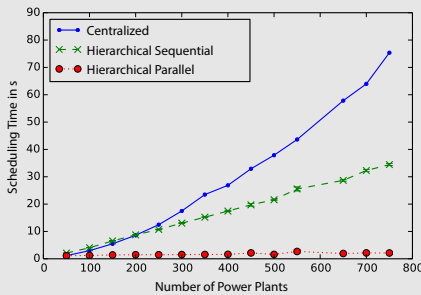


Figure 6.4: Comparison of the time needed to create schedules for different numbers of power plants. “Hierarchical Sequential” corresponds to the time needed to calculate all schedules sequentially, whereas “Hierarchical Parallel” corresponds to the maximum sequential scheduling time. Results are averaged over 2400 data points from 50 drawn sets of plants each considering 48 time steps. Based on data provided in [187].

These results were obtained in our simulation environment for autonomous power management on the basis of predefined system structures. We observed that using small values, such as 5 or 15, for the number of power plants controlled per AVPP leads to fragmented structures where accuracy and efficiency are wasted if the solver used for creating schedules still has capacities for a higher number of power plants. Higher values, such as 35, lead to the best runtime and cost efficiency in all considered cases. Interestingly, the trade-off between cost and runtime performance seems to be regulated by the depth of the hierarchy; deep structures lead to the best runtime performance at slightly higher costs due to more abstractions.

In the following, we explain how such a hierarchy can evolve in a self-organizing manner so that it is able to adapt to the current environmental conditions and the system's internal state. Ultimately, this allows the system to come to a compromise between solution quality and runtime performance itself.

6.1 The Partitioning Problem

In essence, agents form a hierarchy by recursively partitioning an existing subsystem into smaller disjoint subsystems, each of which is represented by an intermediary (that is, an AVPP in our case study). These are then arranged in a tree. For example, we obtain a flat system structure as shown in Figure 6.5b by partitioning all agents below the top-level intermediary of the centralized system depicted in Figure 6.5a. If we now partition the members of the subsystem in the center of the flat system structure into further, say two, subsystems, we get the hierarchy shown in Figure 6.5c.

In all these cases, the agents reorganize the system structure by solving a *partitioning problem* (PP) in which a set $\mathfrak{A} = \{a_1, \dots, a_n\} \subseteq \mathcal{A}$ of $n > 1$ agents a_i is partitioned into non-empty and pairwise disjoint subsets, called *partitions*, that together constitute a *partitioning*. Note that the agents \mathfrak{A} participating in the PP are only a subset of all agents \mathcal{A} in the system and that \mathfrak{A} might also contain intermediaries. In order to form system structures that support the system's goals, the agents \mathfrak{A} must not be distributed arbitrarily among the created partitions. Instead, the composition of the created partitioning is evaluated with respect to a cost function that is based on application-specific formation criteria (see Section 6.3). The costs of the partitioning are to be minimized. Consequently, the PP constitutes a combinatorial optimization problem. For non-trivial cost functions, finding the optimal partitioning is NP-hard.¹ In the power management case study, each partition corresponds to a set of power plants that will be represented by a new AVPP subsequent to the reorganization (see Section 6.4).

We assume that feasible, i.e., valid, partitions are only constrained in terms of a minimum s_{min} and maximum s_{max} size. In the unbounded case (that is, $s_{min} = 1$ and $s_{max} = n$), the number of feasible partitions m grows exponentially with n since there are $m = |\mathcal{P}(\mathfrak{A}) \setminus \emptyset| = 2^n - 1$ feasible *partitions* ($\mathcal{P}(\mathfrak{A})$ denotes the power set of \mathfrak{A}). In this situation, the *Stirling number of the second kind* $\mathcal{S}(n, i)$ represents the number of ways we can partition n agents into a partitioning containing i non-empty partitions (note the exponential growth of $\mathcal{S}(n, i)$ with n) [24]:

$$\mathcal{S}(n, i) = \frac{1}{i!} \cdot \sum_{j=0}^i (-1)^{i-j} \cdot \binom{i}{j} \cdot j^n$$

The total size of the search space, i.e., the total number of possible *partitionings*, is given by the n -th *Bell number* \mathcal{B}_n (e.g., $\mathcal{B}_5 = 52$, $\mathcal{B}_{10} = 115975$, $\mathcal{B}_{50} \approx 1.86 \cdot 10^{47}$, $\mathcal{B}_{100} \approx 4.76 \cdot 10^{115}$). Given that a set of n agents can be partitioned into at most n non-empty partitions, the Bell number can be calculated as the sum of the first n Stirling numbers of the second kind or by using *Dobiński's formula* [33]:

$$\mathcal{B}_n = \sum_{i=1}^n \mathcal{S}(n, i) = \frac{1}{e} \cdot \sum_{k=0}^{\infty} \frac{k^n}{k!}$$

In contrast to the well-known *set partitioning problem* (SPP) (cf. [27]), we suppose that the mere number of feasible partitions prevents us from calculating all of them in advance. In case of the *complete SPP* (cf. [129]), which constitutes the unbounded case, this already needs more than one week on our Xeon machine for small sets consisting of $n = 50$ agents. When taking into account that the set of agents \mathfrak{A} is subject to change over time, pre-calculating all feasible partitions becomes even more impractical.

As opposed to the SPP, we further do not assume that the costs of having a feasible partition included are predefined and additive (note that an additive cost function allows for assessing the partitions')

¹The *set partitioning problem*, which is known to be NP-hard [72], can be reduced to the PP. Section 6.5 discusses the similarities and differences of the set partitioning problem and the PP in detail.

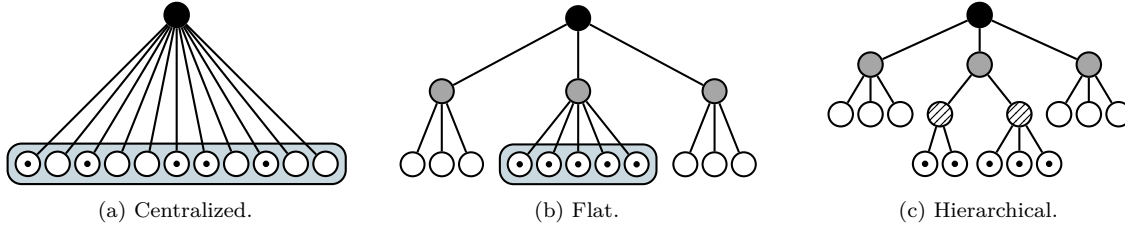


Figure 6.5: By partitioning the participants of a centralized system (see Figure 6.5a), we obtain a flat system structure (see Figure 6.5b). Each created partition embodies a subsystem that is represented by an intermediary (shown in gray). The hierarchical system structure depicted in Figure 6.5c results from recursively partitioning the members of the subsystem in the center of the flat system structure. Again, each created partition is represented by an intermediary (shown as dashed nodes). In both cases, the rectangle indicates the set of agents that participates in the partitioning problem.

quality independently of each other). Instead, we only presume an application-specific cost function that evaluates if a partitioning, i.e., a combination of partitions, is fit for purpose. This allows for flexible objective functions. A more detailed discussion of the differences between the PP and the SPP can be found in Section 6.5.

Many types of agent organizations are based on structures that can be described as a partitioning. If the cost function defines how well agents can work together on a common task, the PP is equivalent to *coalition structure generation* (cf. [199]). If it specifies to group similar or dissimilar agents, the PP is equivalent to *strict partitioning clustering (with outliers)*² (cf. [138]) or *anticlustering* (cf. [203, 215]), respectively. Of course, one can also think of a combination of these heterogeneous objectives.

Before we present an organizational structure that is of particular interest for the class of open systems considered in this thesis in Section 6.3, we extend the PP by partitioning constraints that assist in the creation of scalable system structures in the following section. There, we also provide a formalization of the PP.

6.2 Partitioning Constraints for Guiding Self-Organized Hierarchical Problem Decomposition

As we pointed out at the beginning of this chapter, deep hierarchies – in which, on average, intermediaries control a lower number of agents – can considerably decrease the time needed to create schedules. On the flip side, we noticed that short maximum sequential scheduling times come at the expense of solution quality because deep hierarchies suffer from fragmentation (i.e., many subsystems of small size). The effect of fragmentation mainly manifests in the form of abstraction errors which cause intermediaries to under- or overestimate the capabilities of subordinate intermediaries when creating schedules. While underestimations lead to false or suboptimal decisions which directly worsen the schedules’ quality (e.g., underestimating an AVPP’s price-performance ratio can increase the costs of energy provision), overestimations can be cushioned if the affected subsystem features a sufficient amount of degrees of freedom and redundancy (e.g., multiple similar low-priced power plants). Increasing fragmentation reduces the subsystems’ degrees of freedom as well as redundancy though and thus intensifies the negative effect of overestimations. In the power management case study, an AVPP’s degrees of freedom and scheduling time depend on which and how many dispatchable power plants it controls. Furthermore, since schedules are created top-down (that is, sequentially on each path from the top-level intermediary to a leaf), increasing the hierarchy’s height does not always result in shorter max. seq. scheduling times – even though the average number of agents each intermediary controls declines: This is only the case if the reduction of the max. seq. scheduling time resulting from the lower number of agents an intermediary

²Supported by a separate partition that holds all outliers.

controls outweighs the scheduling times of the additional intermediaries lengthening the sequential path (see Sections 2.2 and 9.2).

As there is a trade-off between the hierarchy's height and the number of agents each intermediary controls, it is necessary to guide the self-organized hierarchical problem decomposition. With regard to the PP, we achieve this by allowing the user or the agents themselves to specify feasible solutions, i.e., valid partitionings, in terms of suitable *ranges* for the number and the size of partitions. These ranges are defined by the minimum n_{min} and the maximum n_{max} number of partitions ($1 \leq n_{min} \leq n_{max} \leq |\mathfrak{A}|$) as well as their minimum s_{min} and maximum s_{max} size ($1 \leq s_{min} \leq s_{max} \leq |\mathfrak{A}|$). In a hierarchical system, the ranges for the number and the size of partitions stipulate how many subsystems or agents should be controlled by an intermediary. Note that both aspects, the size as well as the number of partitions, are needed to come to a compromise between solution quality and performance in terms of runtime: As each partition K in a created partitioning \mathcal{P} represents a new subsystem (i.e., a set of power plants) that is to be controlled by an intermediary (i.e., an AVPP), K 's size influences the complexity of the intermediary's scheduling problems. The number of created partitions, i.e., the size of \mathcal{P} , has an effect on the complexity of the scheduling problem of the superordinate intermediary that governs the created subsystems. Since the boundaries n_{min} , n_{max} , s_{min} , and s_{max} constitute mandatory characteristics of a solution to the PP, they represent hard constraints that we call *partitioning constraints* in the following. The parametrization of the partitioning constraints defines the shape of the self-organizing hierarchy. In terms of the Corridor Enforcing Infrastructure, the conjunction of the partitioning constraints and the constraints specifying the general properties of a partitioning (i.e., each agent must be in exactly one partition) define the borders of a corridor of correct behavior.

Using ranges instead of fixed values provides the system with degrees of freedom needed to optimize the partitionings' composition according to application-specific formation criteria. To form a reasonable hierarchical structure, each partitioning should contain at least $n_{min} \geq 2$ partitions, each of which contains at least $s_{min} \geq 2$ agents. Regarding the example of creating AVPPs, it is required that the size of the resulting AVPPs is not less than two and below a certain threshold that allows them to solve the scheduling problem in adequate time. Analogously, the number of resulting AVPPs must not be below two and below a certain threshold that allows their superordinate AVPP to solve the scheduling problem in adequate time.

Based on the partitioning constraints and an application-specific objective function $f(\mathcal{P})$, the PP as introduced in Section 6.1 can be formalized as follows:

$$\underset{\mathcal{P}}{\text{maximize}} \quad f(\mathcal{P}) \quad (6.2a)$$

$$\text{subject to} \quad \forall a \in \mathfrak{A} : \exists K \in \mathcal{P} : a \in K, \quad (6.2b)$$

$$\forall K, L \in \mathcal{P} : K \neq L \Rightarrow K \cap L = \emptyset, \quad (6.2c)$$

$$\forall K \in \mathcal{P} : s_{min} \leq |K| \leq s_{max}, \quad (6.2d)$$

$$n_{min} \leq |\mathcal{P}| \leq n_{max}, \quad (6.2e)$$

$$\text{with} \quad 1 \leq s_{min} \leq s_{max} \leq |\mathfrak{A}|,$$

$$1 \leq n_{min} \leq n_{max} \leq |\mathfrak{A}|$$

Equations (6.2b) and (6.2c) specify the general properties of a partitioning, that is, each agent $a \in \mathfrak{A}$ participating in the PP must be in exactly one partition. The partitioning constraints are implemented in Equations (6.2d) and (6.2e). As stipulated in Equation (6.2), the PP's objective is to find a partitioning \mathcal{P} at minimal costs, which corresponds to maximizing the objective function $f(\mathcal{P})$.

Note that the partitioning constraints reduce the size of the search space for algorithms that only consider feasible solutions when solving the PP (in this case, the search space corresponds to the solution space). Suitable boundaries can thus lower the time needed to find high-quality solutions. We give empirical evidence of this claim in the evaluation of our PSOPP algorithm (see Section 7.6).

Obviously, as the possible number and size of partitions are interconnected with the number of agents $|\mathfrak{A}|$ to be partitioned, one has to make sure that the problem is not overconstrained. In case of $\mathfrak{A} = \{a_1, a_2, a_3\}$, e.g., there is no feasible solution if we set n_{min} and s_{min} to two since we would have to

create at least two partitions of size two each. Either n_{min} or s_{min} would have to be relaxed, i.e., set to 1. The following definition specifies valid parametrizations of the partitioning constraints dependent on the set of agents \mathfrak{A} to be partitioned.

Definition 2 (Valid Parametrization of the Partitioning Constraints). *With respect to a set of agents \mathfrak{A} participating in the partitioning problem, a parametrization $s_{min}, s_{max}, n_{min}, n_{max}$ of the partitioning constraints is **valid** if and only if the following conjunction of four conditions is satisfied:*

$$\text{canSatisfyPartitioningConstraints}(\mathfrak{A}) :\Leftrightarrow \quad (6.3)$$

$$|\mathfrak{A}| \geq n_{min} \cdot s_{min} \wedge |\mathfrak{A}| \leq n_{max} \cdot s_{max} \wedge |\mathfrak{A}| \geq \left\lceil \frac{|\mathfrak{A}|}{s_{max}} \right\rceil \cdot s_{min} \wedge |\mathfrak{A}| \leq \left\lfloor \frac{|\mathfrak{A}|}{s_{min}} \right\rfloor \cdot s_{max}$$

The first condition states that the partitions' minimum number has to be low and their minimum size small enough so that the participating agents suffice to create a valid partitioning. The second condition states that the partitions' maximum number has to be high and their maximum size large enough so that all agents can be assigned to a partition. The third condition states that the number of agents to partition must be sufficient to fill the minimum number of necessary partitions $\left\lceil \frac{|\mathfrak{A}|}{s_{max}} \right\rceil$ up to the required minimum s_{min} . The fourth condition states that the maximal number of partitions $\left\lfloor \frac{|\mathfrak{A}|}{s_{min}} \right\rfloor$ that can be created with the given set of agents must be sufficient to assign each agent to a partition without exceeding the partition's maximum size s_{max} .

For example, if $|\mathfrak{A}| = 9$ and $s_{min} = 4, s_{max} = 4, n_{min} = 2, n_{max} = 5$, the first and the second condition hold, but the third and the fourth condition are not satisfied because $9 \not\geq \left\lceil \frac{9}{4} \right\rceil \cdot 4 = 12$ and $9 \not\leq \left\lfloor \frac{9}{4} \right\rfloor \cdot 4 = 8$, respectively. We can satisfy all conditions by changing $s_{min} = 4$ to $s_{min} = 3$.

Whenever the system solves an instance of the PP, it has to specify a parametrization of the partitioning constraints that satisfies Equation (6.3).

6.3 Homogeneous Partitioning to Promote System Stability and Robust Resource Allocation

Because the demand usually exceeds the limited resources of a single agent, the agents have to solve the resource allocation problem in cooperation. In the literature, there is a multitude of organizational paradigms promoting a system's scalability, cooperation, and efficiency (Horling and Lesser [92] provide an overview). While many of these structures, such as *coalitions* [185] or *teams* [30], can be represented as a partitioning, their specific characteristics are decisive factors when it comes to the system's and the agents' ability to achieve their goals in a concrete setting.

Coalitions are a concept known from the field of game theory [180]. They enable cooperation between self-interested and individually rational agents. They are goal-directed in the sense that they are created in order to accomplish one or more specific tasks. Because they dissolve after these tasks have been performed, they are rather short-lived. While coalitions do not always have to be disjoint [199], approaches to coalition formation have in common that they strive for the coalition structure of maximal utility. To this end, a characteristic function is used that assigns a value to each possible coalition in accordance with the tasks to be performed. In case of disjoint coalitions, a coalition structure corresponds to a partitioning. With respect to open systems, literature often regards the problem of participants that might not only show self-interested behavior but also intentional misbehavior by lying about their capabilities, the utility of performing a task, etc. [59]. Since suitable coalition structures depend on the agents' promised contributions, the system has to make sure that these promises are kept or to form coalitions of agents that mutually trust each other. While the former can be achieved by employing an incentive-compatible mechanism as discussed in more detail in Chapter 13, the latter has been studied by incorporating trust relationships into the valuation of coalitions [43].

In contrast to coalitions, teams consist of cooperative agents that pursue a common goal over a longer period of time [92]. Especially in open systems, the challenge is to create teams that can persist despite changes in the agents' behavior, the environmental conditions, and the uncertainties the system is exposed to. Given that cooperation is likely to be most beneficial and least uncertain with trustworthy agents, it has been proposed to form teams of agents that mutually trust each other, such as *clans* [77]. Untrustworthy agents are marginalized by excluding them from most interactions. Klejnowski [117] introduced a similar concept to clans for the domain of Desktop Grid Computing, called *Explicit Trusted Communities* (eTCs). The main difference to clans and coalitions is that each eTC is represented by an explicit manager which administrates memberships, deals with conflicts, and governs the participating agents with norms. Clans and eTCs incentivize untrustworthy agents to change their behavior by preferring interactions with trustworthy agents (or even restricting them to these agents). Ultimately, forming teams of mutually trusting agents aims at a more efficient and robust system – at least with regard to the members of clans or eTCs. While these types of organizations are not necessarily limited to intentional misbehavior, the formation of exclusive groups assumes that agents can be excluded from other parts of the system without jeopardizing the overall system's stability and efficiency. In the following, we explain why this is not a feasible approach to solve our resource allocation problem and present an alternative organizational paradigm afterwards.

Homogeneous Partitioning

The class of open systems regarded in this thesis suffers from agents that, on the one hand, introduce uncertainties and thus jeopardize the systems' stability but, on the other hand, cannot or should not be excluded from participating in the resource allocation problem. In the power management case study, for instance, the uncertainty of the future output could be significantly reduced if it was possible to turn off all weather-dependent power plants. However, this is not feasible because the system might depend on their output at on-peak hours and because their low-cost generation improves the system's economic efficiency. Concerning the latter, legal regulations, such as the German Renewable Energy Act, further incentivize the utilization of renewable energy sources. If, in such a situation, scalability demands that the agents self-organize into multiple subsystems, the system's stability and efficiency ultimately hinges on the form of organization which the agents establish on the basis of their trustworthiness and other criteria that have an effect on efficiency and robustness against disturbances. So the question that remains is which objective function $f(\mathcal{P})$ the agents should use to evaluate the quality of a partitioning \mathcal{P} when solving the PP (see Equation (6.2a)). An important factor influencing this decision is that our approach to problem decomposition is grounded on subsystems that pursue the same goal as the overall system, that is, to fulfill (a part of) the overall demand. Another factor is that a reorganization changes the agents' interaction partners, which requires the agents to build new probabilistic models of their behavior, e.g., in the form of trust values or Trust-Based Scenarios (see Part II). As frequent changes might temporarily impair the models' accuracy and thus the system's efficiency and robustness, the organizational structure should be rather stable. Summarizing, the challenge is to create a long-lasting organizational structure that enables each subsystem to satisfy its fraction of the overall demand although the circumstances (i.e., the actual demand, the actual available resources, and the degree of uncertainty) under which this goal has to be achieved are unknown at the time the structure is formed. Note that a single subsystem not being able to cope with its uncertainties endangers the goal of the overall system.

In case of strict *partitioning clustering*, structures consisting of groups of *similar elements* (here, agents) can be obtained by minimizing the sum of the squared Euclidean distances between the agents' properties p_a (e.g., a power plant's rate of change) and their cluster center, i.e., the centroid, of the partition $K \in \mathcal{P}$ the agents $a \in \mathcal{A}$ belongs to. The centroid is defined as the mean $p_\mu(K) = \frac{1}{|K|} \cdot \sum_{a \in K} p_a$ of the properties of the partition members (e.g., the mean rate of change of the power plants in partition K). In this case, the objective function in Equation (6.2a) thus corresponds to the negative value of the "classic" k-means distance measure $f_C(\mathcal{P})$ ($\|\cdot\|$ denotes the Euclidean norm) [138]:

$$f_C(\mathcal{P}) = - \sum_{K \in \mathcal{P}} \sum_{a \in K} \|p_a - p_\mu(K)\|^2 \quad (6.4)$$

According to Späth [203], $f_C(\mathcal{P})$ is equivalent to $\sum_{K \in \mathcal{P}} |K| \cdot \|p_\mu(K) - p_\mu(\mathfrak{A})\|^2$, where $p_\mu(\mathfrak{A})$ represents the mean of the properties of all agents participating in the PP. Clustering thus yields dissimilar partitions consisting of similar agents. The drawback of a structure of dissimilar partitions is that some subsystems are likely to depend on the capabilities of others to be able to fulfill their fraction of the overall demand. Imagine, for instance, an AVPP consisting of well-predictable dispatchable power plants and another AVPP of non-dispatchable power plants whose output is subject to uncertain weather conditions. To hold the balance between power supply and demand, the AVPP of weather-dependent power plants depends on reactive supply adjustments of the AVPP of dispatchable power plants. Such a situation not only impairs the quality of the solution to the underlying resource allocation problem (e.g., because costly peaking power plants have to be ramped up) but also puts the goal of balancing supply and demand at risk (e.g., if the available adjustable output of the AVPP of dispatchable power plants is insufficient to compensate for a deviation between production and consumption because it did not incorporate the possible deviations of the AVPP of weather-dependent power plants into the schedule creation). In order to increase robustness, the AVPP of dispatchable power plants would have to anticipate and incorporate uncertainties of the other AVPP into its schedule creation. This dependency evidently contradicts the idea of problem decomposition, though. Hence, another type of organization is required to deal with uncertain system participants.

For situations in which untrustworthy agents cannot or should not be excluded from the system, we propose the concept of *homogeneous partitioning*. It aims at the creation of organizations that are, with respect to certain criteria, as similar as possible (recall that each organization pursues the same goal). For instance, we want to establish subsystems that feature a similar ratio between uncertainty (i.e., predictability) and degrees of freedom in terms of controllability. That is because each subsystem must be (equally) able to compensate for local deviations between supply and demand (e.g., for each AVPP, the ratio between the sum of typical variations in prediction errors of its weather-dependent power plants and the sum of the rate of change of its dispatchable power plants should be similar). This idea is based on the assumption that a centralized system imposes an upper bound on this ratio and the fact that similar partitions exhibit properties that are similar to those of the set of agents they are based on: The global knowledge about possible uncertainties and the centralized control over all dispatchable agents allow the system to fulfill its task as well as possible. If each organization now exhibits similar characteristics, such as a similar ratio between uncertainty and controllability, they approximate the corresponding ratio of the centralized system. Consequently, they also inherit its positive properties. Economic aspects further call for subsystems that can provide resources in a cost-efficient way (e.g., the AVPPs' average costs per kWh should be similar). Because the performance of the schedule creation is defined by the maximum sequential scheduling time, the scheduling times of the subsystems that result from partitioning the agents \mathfrak{A} should be similar. In the case study, this can be achieved by creating AVPPs with a similar number of dispatchable power plants. All these examples of formation criteria aim at forming similar partitions, i.e., subsystems. Due to the system's heterogeneity, these similar partitions are likely to be composed of dissimilar agents.

In *homogeneous partitioning*, we are interested in minimizing the dispersion of the partitions $K \in \mathcal{P}$ with regard to a specific property p_K , such as the ratio between uncertainty and controllability, the mean costs, or the number of dispatchable power plants. We achieve this by using the negative uncorrected sample standard deviation $f_{HP}(\mathcal{P})$ as objective function in Equation (6.2a) (with $n = |\mathfrak{A}|$):

$$f_{HP}(\mathcal{P}) = -\sqrt{\frac{1}{n} \cdot \sum_{K \in \mathcal{P}} (p_K - p_\mu(\mathcal{P}))^2} \quad (6.5)$$

Here, $p_\mu(\mathcal{P})$ denotes the mean value of the property p_K over all partitions $K \in \mathcal{P}$. Note that, in contrast to clustering, homogeneous partitioning specifies the desired partitioning on the basis of the partitions' *relative* composition instead of their *individual* composition. That is, in homogeneous partitioning, the quality of the partitioning \mathcal{P} depends on the composition of partitions $K \in \mathcal{P}$ in relation to the composition of the other partitions in \mathcal{P} . Among other things, a property p_K of a single partition can be the *sum* $p_K = \sum_{a \in K} p_a$ (e.g., the number of dispatchable power plants per partition K) or the *mean* $p_K = \sum_{a \in K} p_a / |K|$ (e.g., similar production costs per partition K) of the corresponding property of its

members. As stated above, there are situations in which it is useful to employ a *ratio* $p_K = \phi(K)/\phi'(K)$ between two properties $\phi(K), \phi'(K)$ aggregated over K 's members to specify a certain characteristic of K (e.g., the ratio between accumulated uncertainty and controllability per partition K). In the following, we refer to these *modes* as homogeneous partitioning sum, mean, and ratio, respectively, according to the aggregation function in use.

An important attribute of homogeneous partitioning is that the resulting subsystems are loosely coupled, meaning that the scheduling problem is decomposed into independent sub-problems: Since each organization has a similar ratio of uncertainty to controllability, an organization's dispatchable agents can utilize their degrees of freedom to compensate for the uncertainties stemming from untrustworthy agents internally. This avoids affecting other organizations or involving them into the decision-making process.³ As a result, the system's robustness and efficiency increases. This is why homogeneous partitioning should be preferred to other types of organizations, such as those consisting of homogeneous agents, for the class of open systems considered in this thesis.

A related approach that targets a similar system class has been presented by Chalkiadakis et al. [48]. Their central idea is to mitigate unintentional uncertainties originating from weather-dependent power plants by forming coalitions in a way that these uncertainties cancel each other out (e.g., by combining a solar power plant that overestimates its output with a solar power plant that underestimates its output). However, their approach does not aim for long-living subsystems that are created for the purpose of decomposing a resource allocation problem that has to be solved in cooperation. As a consequence, they neglect the degrees of freedom of dispatchable agents as formation criterion, among others. While this limits the applicability of their approach to our resource allocation problem, it might still be beneficial to combine their idea of reducing uncertainties with our idea of creating partitions featuring a similar ratio of uncertainty to controllability.

An overview of the formation criteria for AVPPs is provided at the end of this section. Since we combine multiple different formation criteria, the power plants have to solve a multi-objective combinatorial optimization problem when organizing into AVPPs. In this case, the PP's objective function is a combination of several objectives (e.g., a convex combination). If an AVPP observes that one or more of these criteria are not sufficiently satisfied within its partitioning, it triggers a reorganization (see Section 6.4).

Comparison of Homogeneous Partitioning to Anticlustering

Späth [203] and Valev [215] proposed the concept of anticlustering as the opposite of obtaining a clustering according to the k-means distance measure given in Equation (6.4):

$$f_{AC}(\mathcal{P}) = \sum_{K \in \mathcal{P}} \sum_{a \in K} \|p_a - p_\mu(K)\|^2 \quad (6.6)$$

Anticlustering thus aims for partitions consisting of dissimilar elements (here, agents). Because maximizing $f_{AC}(\mathcal{P})$ is equivalent to maximizing $-\sum_{K \in \mathcal{P}} |K| \cdot \|p_\mu(K) - p_\mu(\mathcal{A})\|^2$ (cf. [203]), clustering dissimilar agents yields partitions whose mean properties $p_\mu(K)$ correspond to the mean properties $p_\mu(\mathcal{A})$ of all agents participating in the PP [203]. At first glance, it thus seems as if anticlustering is the same as homogeneous partitioning *mean*. However, it turns out that this is only true for *optimal* anticlusterings. In fact, the anticlustering metric implies another order on candidate solutions than homogeneous partitioning mean. This is shown by the following example in which we partition a set of four power plants with respect to their maximum output. For illustration purposes, we assume that each power plant's maximum output is unique. This allows us to represent partitionings of the power plants by partitionings of their maximum outputs $\{4, 6, 7, 9\}$. Our goal is to form partitions, i.e., AVPPs, with similar mean maximum outputs. Let us assume that an optimizer finds the two candidate solutions $\mathcal{P} = \{\{4, 6\}, \{7, 9\}\}$ and $\mathcal{Q} = \{\{4\}, \{6, 7, 9\}\}$. When employing anticlustering, the values of the objective function for \mathcal{P} and \mathcal{Q} are -9.0 and ≈ -8.3 , respectively. So, according to anticlustering, \mathcal{Q} is better

³In some situations, an organization might still have to consult other organizations to compensate for high deviations.

than \mathcal{P} . Homogeneous partitioning mean, however, prefers \mathcal{P} as its value of the objective function is -1.5 as opposed to ≈ -1.7 in case of \mathcal{Q} . In fact, only the order induced by homogeneous partitioning mean complies with our intention. This becomes evident when regarding the absolute difference between the partitions' mean values, which is 3.0 for \mathcal{P} instead of ≈ 3.3 for \mathcal{Q} .

As the large search space prevents us from taking optimal results for granted, this example illustrates that anticlustering should not be used in place of homogeneous partitioning mean. Furthermore, the general concept of homogeneous partitioning is not limited to establishing similar mean values. Instead of forming AVPPs with similar mean maximum outputs, we could thus also form AVPPs with similar total outputs. This cannot be achieved with anticlustering.

Criteria for the Formation of AVPPs

With regard to the power management case study, our main concern is to maintain a structure of AVPPs that adequately decomposes the scheduling problem and, at the same time, allows each AVPP to locally deal with uncertainties stemming from non-dispatchable power plants. Based on these aspects, we define the following criteria for forming appropriate AVPPs (this list is not intended to be exhaustive).

Homogeneous Partitioning Ratio:

- *Similar Ratio of Uncertainty to Controllability:* To ensure that AVPPs are equally able to deal with local uncertainties, AVPPs should feature a similar ratio of the expected variation in the deviations between the actual and the expected output of their subordinate non-dispatchable power plants to their total adjustable output per minute. The former results from the variance of the quality of a power plant's output predictions. This type of uncertainty is captured in a power plant's *predictability*, which is part of the basic trust model we defined in Section 4.2.
- *Similar Ratio of Dispatchable to Non-Dispatchable Output:* To further deal with uncertainties resulting from non-dispatchable power plants, each AVPP should feature a similar ratio of its total dispatchable to its total non-dispatchable output.

Homogeneous Partitioning Sum:

- *Similar Number of Dispatchable Power Plants:* To equalize the time needed to create schedules among the AVPPs, each AVPP should control a similar number of dispatchable power plants (including subordinate AVPPs).

Homogeneous Partitioning Mean:

- *Similar Average Production Costs:* To avoid that some AVPPs cause high costs when providing energy (even if an AVPP's schedule prescribes that its expensive dispatchable power plants should be turned off, it might still have to ramp some of them up in order to compensate for fluctuation in its local residual load), the average unit price of electricity should be similar.

Clustering and Anticlustering:

Besides these properties targeting the power plants' physical properties and their predictability, their geographic location should also be considered when forming AVPPs. Here, we opt for either anticlustering or clustering, depending on whether we abstract from the characteristics of the underlying power grid or not:

- *Anticlustering – Geographic Distribution:* If we abstract from the power grid and its different voltage levels and only regard the mere balance of produced and consumed power, a power system with a high percentage of weather-dependent output can profit from creating AVPPs consisting of power plants with a high geographic distribution. Due to local weather conditions, combining power plants that reside in different geographic locations can reduce uncertainties stemming from aggregation effects. For instance, an unexpected lower output of a wind turbine in northern Germany might be cushioned by a solar power plant in southern Germany whose output is higher than expected. In such a situation, the AVPP might be able to save expenses resulting from reactive output adjustments of costly peaking power plants.
- *Clustering – Geographic Proximity:* When taking the characteristics of the underlying physical infrastructure, such as the power grid's different voltage levels [116], into account, AVPPs should

be composed of nearby power plants, e.g., those that are connected to the same or close distribution networks as power flow between different distribution networks is often inefficient or technically infeasible nowadays.

These examples show that, with regard to specific formation criteria, homogeneous partitioning makes statements about the relative composition of AVPPs, whereas clustering and anticlustering make statements about the composition of individual AVPPs.

Homogeneous Partitioning Yields Stable System Structures

We hypothesize that, in situations in which the agents' behavior is influenced by an uncertain and highly dynamic environment, homogeneous partitioning yields more stable organizational structures than clustering. We base our hypothesis on the definition of clustering and homogeneous partitioning provided in Equations (6.4) and (6.5): In general, environmental changes are likely to alter the agents' behavior in different ways. Considering clusters of similar agents, these changes cause their members to become too dissimilar. Regular reorganizations are required to maintain an adequate clustering. Regarding homogeneous partitioning, the organizations remain rather similar because several of them are affected by the environmental changes; this is why the need for reorganizations is lower than in case of clustering.

To give empirical evidence of our claim, we evaluated the stability of AVPPs that are formed in systems consisting of different numbers $n \in \{250, 500, 1000, 2000, 4000\}$ of power plants. Each of these power plants has a trust value that reflects the accuracy of its output predictions. In our evaluation, environmental changes correspond to changing weather conditions that were modeled on the basis of a Markov chain. Depending on the type of power plant, these changes influence the quality of the power plants' output predictions and, in turn, its trust value in different ways.

To investigate the stability of the organizational structures resulting from homogeneous partitioning and clustering, we ran experiments in which the power plants formed (1) homogeneous partitionings in terms of partitions with similar mean trust values (objective HPm) as well as (2) clusters of power plants with similar trust values according to the k-means distance measure (objective C).

For each combination of system size n and objective function, we ran 1000 experiments. In each experiment, we used a randomly generated partitioning as initial system structure and simulated 300 time steps. In every time step, the environment changed and influenced the power plants' trust values accordingly. To detect the need for a reorganization, the power plants further evaluated the quality of their current organizational structure, i.e., partitioning \mathcal{P} , by means of the fitness function $f_{\text{HPm}}(\mathcal{P})$ or $f_{\text{C}}(\mathcal{P})$, depending on whether objective HPm or C was employed. These fitness functions correspond to the objective function in Equation (6.2a). Based on a normalized fitness value between 0 and 1 that was obtained by means of empirically identified best and worst fitness values, the power plants decided to reorganize the partitioning if its normalized fitness was below 0.9. Otherwise, they left the structure unchanged. Reorganizations were performed by our partitioning algorithm PSOPP, which is presented in Chapter 7 (parameters were chosen according to the results of our parameter search outlined in Section 7.6). Each reorganization was restricted to 3 s. To minimize the influence of the partitioning constraints on our results, we used $s_{\min} = n_{\min} = 2$ and $s_{\max} = n_{\max} = n$ as parametrization. As we are interested in the stability of partitionings, we restricted the formation of AVPPs to a flat system structure (cf. Figure 6.5b). In total, this amounts to 10000 experiments.

As we proposed in [64], we use the *WAT* metric [103] to measure the stability of organizational structures. This metric has been devised for self-adaptive systems and measures the performance of a self-adaptation mechanism as the ratio of *working time* to *adaptivity time*:

$$WAT := \frac{\text{working time}}{\text{adaptivity time}}$$

Consequently, the higher the *WAT* value, the better the performance of the employed self-adaptation mechanism. The *WAT* metric is based on the idea that a self-adaptation mechanism should keep the

#Agents	objective HPm					objective C				
	250	500	1000	2000	4000	250	500	1000	2000	4000
WAT per Run	237.85 (75.81)	270.75 (58.12)	287.15 (39.00)	212.78 (81.65)	125.53 (74.19)	92.86 (31.86)	85.67 (13.10)	71.24 (10.11)	19.49 (6.12)	2.73 (0.75)
#Reorganizations per Run	1.42 (0.56)	1.17 (0.41)	1.07 (0.26)	1.65 (0.70)	3.05 (1.40)	3.35 (0.65)	3.50 (0.54)	4.18 (0.60)	14.86 (2.75)	72.05 (11.02)
No. of Partitions per Time Step	3.12 (1.39)	3.10 (0.57)	3.21 (0.60)	47.71 (114.23)	196.68 (279.14)	121.68 (1.31)	242.75 (2.56)	479.33 (9.01)	951.90 (25.11)	1912.80 (65.32)
Size of Partitions per Time Step	80.05 (36.92)	161.08 (69.64)	311.29 (144.64)	41.92 (154.44)	20.34 (119.67)	2.05 (0.23)	2.06 (0.24)	2.09 (0.28)	2.10 (0.30)	2.09 (0.30)
Normalized Fitness per Time Step	0.96 (0.02)	0.96 (0.02)	0.96 (0.02)	0.97 (0.02)	0.97 (0.02)	0.97 (0.01)	0.96 (0.01)	0.95 (0.01)	0.92 (0.00)	0.88 (0.01)

Table 6.1: Evaluation results for the objectives homogeneous partitioning (HPm) and clustering (C) for different system sizes. All values are averages over 1000 experiments consisting of 300 time steps each; values in parentheses denote standard deviations. A reorganization was triggered as soon as the normalized fitness was below 0.9.

system in an adequate working state with as little disruption as possible.

Here, we apply the *WAT* metric to evaluate the stability of an organizational structure. To this end, a time step that did not require a reorganization increased the *working time* by one, whereas one that needed a reorganization incremented the *adaptivity time* by one. As each experiment comprised 300 time steps, the *WAT* value is from the interval $[0, 300]$, where $WAT = 0$ states that the system spent all time in reorganization, and $WAT = 300$ means that the system never had to change its structure.

As expected, system structures based on homogeneous partitioning HPm are far more stable than those using clustering C (see Table 6.1). This is not only reflected in the mean *WAT* values but also in the mean number of reorganizations per experiment. For $n = 250$, for instance, C achieves only 39% of HPm's *WAT* value. Accordingly, C needs more than twice of HPm's number of reorganizations to ensure the partitioning's required minimum fitness of 0.9. For $n = 4000$, C obtains only 2% of HPm's *WAT* value. In this case, C needs about 23 times the number of reorganizations HPm required to maintain the partitioning's quality.

Interestingly, the *WAT* value slightly increases for HPm from $n = 250$ to $n = 1000$. Together with the observation that the mean number of partitions is more or less constant for $n \in \{250, 500, 1000\}$ and that the mean size of partitions grows accordingly, this indicates that homogeneous partitionings consisting of large partitions contribute to the organizational structures' stability. This also explains why the structures' stability declines with $n > 1000$: Here, PSOPP was not able to obtain structures of the same quality as for $n \leq 1000$, which is mirrored in the abrupt decrease of the size of partitions.⁴ That is because we did not increase the time limit for reorganizations with n . As the mean size of partitions shrinks with $n > 1000$, also the *WAT* value and thus the structures' stability deteriorates. To promote the structures' stability for $n > 1000$, we could (1) increase the time limit for reorganizations with n , or (2) lower the complexity of forming adequate partitionings by allowing the agents to establish hierarchical system structures in which the PP is recursively solved in each subsystem. The latter possibility is explained in Chapter 9, where we use our definition of the PP and the partitioning algorithms PSOPP and SPADA (see Chapters 7 and 8) to enable self-organizing hierarchies.

With regard to C, the *WAT* value significantly declines with the system size n . This is accompanied by an increase in the number of reorganizations. Although the partitionings' basic characteristics in terms of the size of partitions do not change notably with n , we observe that, the larger the system's size, the more the agents struggle with preserving the partitioning's quality measured in the form of the normalized fitness. In case of $n = 4000$, the partitioning's mean quality achieved with C is even below the demanded minimum of 0.9. This indicates that it is also harder to obtain high-quality clusterings than high-quality homogeneous partitionings (our results presented in Section 7.6 support this hypothesis).

Summarizing, these results not only highlight that structures based on homogeneous partitioning yield far more stable system structures than clustering, but also that the structures' stability benefits from large organizations.

⁴The slight increase of the normalized fitness value seems to contradict this statement. However, this can be attributed to the fact that it was also more difficult to find empirically worst fitness values used for normalization.

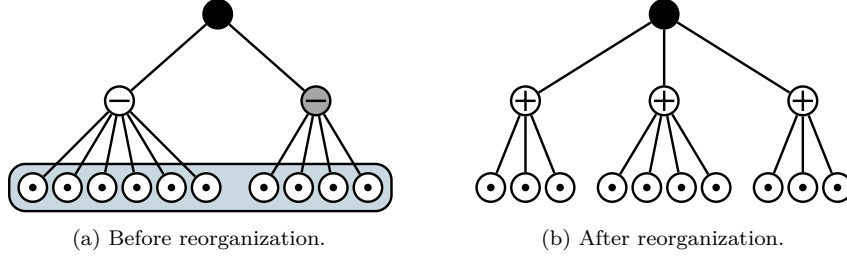


Figure 6.6: Reorganization of a flat system structure: The initiating intermediary is marked in gray. The agents \mathfrak{A} participating in the re-partitioning are highlighted by the rectangle. After the reorganization, the two intermediaries “ \ominus ” standing for the former partitioning \mathcal{P} are replaced by the three new intermediaries “ \oplus ” representing the partitions of the new partitioning \mathcal{P}' .

6.4 Initiating the Reorganization of a Partitioning and Adopting the Reorganization Result

To recognize the need for reorganizing a partitioning, each intermediary periodically evaluates if the partitioning it resides in left the corridor of correct behavior that is specified by a so-called *composition constraint*. The composition constraint is a conjunction of the partitioning constraints (see Section 6.2), the constraints defining the general properties of a partitioning, and another set of constraints used to ensure the quality of the partitioning with regard to application-specific criteria. To comply with the idea of the Restore Invariant Approach (see Section 1.4), one has to make sure that the application-specific constraints either correspond to or are implied by the formation criteria used in the PP’s objective function. The partitioning constraints can only be violated when one or more agents entered or left the system so that the size or the number of subsystems, i.e., partitions, in the partitioning \mathcal{P} became invalid.⁵ An application-specific constraint aiming at preserving the partitionings’ quality is violated if the quality of the partitioning falls below a given threshold. This can be the result of environmental changes, changing agent behavior, or changes in the set of system participants. As for AVPPs, the application-specific constraints mainly correspond to the formation criteria listed in Section 6.3. Additionally, AVPPs trigger a reorganization if the scheduling times of the AVPPs constituting a partitioning are too dissimilar. This reduces the necessity of increasing the height of the hierarchy (see Section 9.2).

In case of a violation of the composition constraint, the intermediary observing the violation triggers a reorganization of the partitioning \mathcal{P} it is situated in. All steps the reorganization comprises are carried out by the controller part of the Corridor Enforcing Infrastructure (see Section 1.4). The goal of the reorganization is to bring the system back into the corridor of correct behavior, that is, to re-establish the satisfaction of the composition constraint. Here, a reorganization corresponds to a *re-partitioning* of the set of agents $\mathfrak{A} \subseteq \mathcal{A}$ constituting \mathcal{P} . Of course, the self-organization algorithm that is used to re-partition \mathfrak{A} can only come up with a feasible solution if the parametrization of the partitioning constraints is valid, i.e., if Equation (6.3) holds for \mathfrak{A} . Otherwise, the PP would be infeasible. The intermediary initiating the reorganization therefore has to make sure that an appropriate parametrization of the partitioning constraints is available. Afterwards, the intermediary initializes the self-organization algorithm with the current partitioning \mathcal{P} . This allows the algorithm to conduct the re-partitioning by making selective changes or utilizing \mathcal{P} as a source of information about (un)suitable partitions, for instance.

After the partitioning algorithm terminated, the result \mathcal{P}' is adopted by updating the reorganized part of the system structure. Basically, the update is performed by replacing the intermediaries standing for the partitions in the former partitioning \mathcal{P} by new intermediaries representing the partitions in \mathcal{P}' (depending on how much \mathcal{P}' differs from \mathcal{P} , it can be useful to reuse existing intermediaries instead of replacing all of them). Each agent $a \in \mathfrak{A}$ is then assigned to its new intermediary. This intermediary

⁵We assume that each agent entering the system becomes a member of an arbitrary existing partition.

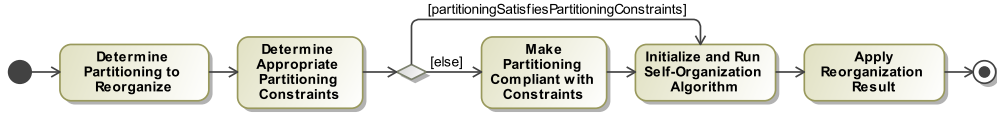


Figure 6.7: The single steps performed by a controller to reorganize a partitioning. The step “Make Partitioning Compliant with Constraints” is only needed if agents entered or left the system in such a way that the partitioning constraints are violated.

corresponds to the partition containing a . Figure 6.6 illustrates this process for an exemplary flat system structure. In such a structure, the set of agents \mathfrak{A} to be re-partitioned consists of all leaves. In Chapter 9, we thoroughly explain when which parts of hierarchical system structures have to be reorganized. Figure 6.7 summarizes the different steps performed in the course of a re-partitioning.

If a reorganization has been triggered because \mathcal{P} does not abide by the partitioning constraints any more, the initiating intermediary re-establishes \mathcal{P} ’s compliance before it initializes the partitioning algorithm. Because the partitioning algorithm is thus always provided with a valid partitioning, we restrict its responsibility to optimizing \mathcal{P} ’s composition with regard to the application-specific formation criteria within the boundaries of the partitioning constraints.

Making the Partitioning to be Reorganized Compliant with the Partitioning Constraints

To correct a partitioning \mathcal{P} that currently does not satisfy the partitioning constraints with a *minimal number of changes*, the initiating intermediary first determines the average number of agents $x = \frac{|\mathfrak{A}|}{|\mathcal{P}|}$ per partition in \mathcal{P} . The number x indicates whether \mathcal{P} contains too few or too many partitions, or if it is sufficient to enlarge too small or to shrink too big partitions by changing the affiliation of some agents. Depending on x , the intermediary proceeds as follows:

1. **Too small or too big partitions ($s_{min} \leq x \leq s_{max}$):**

For each too large partition $K \in \mathcal{P}$, iteratively move $|K| - s_{max}$ agents to the smallest partition $L \in \mathcal{P}$ in this iteration. Since Equation (6.3) holds, it is guaranteed that L is not yet filled to capacity, i.e., $|L| < s_{max}$.

For each too small partition $M \in \mathcal{P}$, iteratively integrate $s_{min} - |M|$ agents from the largest partition $N \in \mathcal{P}$ in this iteration. Since Equation (6.3) holds, it is guaranteed that N is large enough, i.e., $|N| > s_{min}$.

2. **Too few partitions ($x > s_{max}$):**

Create $\left\lceil \frac{|\mathfrak{A}|}{s_{max}} \right\rceil - |\mathcal{P}|$ initially empty partitions that are successively filled up with agents as described in case 1 “Too small or too big partitions”. Note that this is the minimum number of partitions that has to be created. Since Equation (6.3) holds, it is guaranteed that each partition contains at least s_{min} and no more than s_{max} agents.

3. **Too many partitions ($x < s_{min}$):**

Dissolve $|\mathcal{P}| - \left\lfloor \frac{|\mathfrak{A}|}{s_{min}} \right\rfloor$ partitions by moving their members to other partitions in \mathcal{P} . Afterwards, correct the size of too large partitions as described in case 1 “Too small or too big partitions”. Note that this is the minimum number of partitions that has to be dissolved. Since Equation (6.3) holds, it is guaranteed that each partition contains at least s_{min} and no more than s_{max} agents.

Please note that the same procedure can be used to create an initial partitioning satisfying the partitioning constraints when bootstrapping the system. In this case, the procedure is carried out for the grand coalition $\mathcal{P} = \{\mathfrak{A}\}$ or another arbitrary partitioning.

In Chapters 7 and 8, we introduce two partitioning algorithms that enable self-organizing system structures that rely on the general concept of partitionings. Afterwards, we explain the self-organized

formation of hierarchical system structures on the basis of these algorithms in Chapter 9. While we considered the reorganization of a specific partitioning within such a hierarchy in this section, Chapter 9 also provides deeper insights into the matter when which of its parts have to be reorganized, including control actions that can increase or decrease the height of the hierarchy.

6.5 Related Work

In this section, we discuss related work in the context of self-organizing hierarchical system structures, hierarchical problem decomposition, and the PP in more detail. We start with an overview of different models for hierarchical organizations, approaches to hierarchical problem decomposition, and algorithms for the dynamic formation of hierarchies. As our self-organizing hierarchy results from recursively solving the PP, we outline problems related to the PP, point to relevant application domains, and review corresponding algorithms from the body of literature afterwards.

Hierarchical System Structures and Hierarchical Problem Decomposition

In numerous MAS, a crucial step is to establish an organizational structure that supports the agents' and the system's objectives. Horling and Lesser [92] discuss a number of different organizational paradigms that promote scalability, cooperation, and efficiency in MAS. Most importantly, they identify *hierarchies* and *holarchies* as the main approaches to deal with complexity and scalability issues. We have published parts of the following discussion of hierarchies, holarchies, and systems of systems in [207].

Holarchies are a special kind of hierarchical organization. The concept of a *holon*, originally described by Koestler [120], refers to a recursive, self-similar structure. Each holon either represents a single agent (e.g., a physical power plant) or a collective (e.g., an AVPP). A *holarchy* is a hierarchy of holons. In modern uses of this concept, e.g., for holonic manufacturing systems, a holarchy is defined as a system of such holons cooperating to achieve a common goal [53]. A holarchy used in this context is usually not changed at runtime but *predefined* by the designer to meet the specific system requirements. Frey et al. [69] introduce a holonic architecture for smart microgrids. Their work does not provide a solution to the problem of forming such structures at runtime, though. The algorithms introduced in this thesis (see Chapters 7 to 9) can be used to create special instances of the proposed architecture, viz., hierarchies, in a self-organizing manner.

Holarchies and hierarchies are directly related to *systems of systems* (SoS). SoS are composed of systems that are themselves complex systems. They are usually distributed in nature and of large scale [125]. A lot of work on SoS originates in a military context (see, e.g., [140]) where the interconnection of different complex systems is a must to provide battlefield information and control of a wide array of weapon systems and sensors. Although these systems are heavily connected, they remain independent in many ways. Key characteristics of SoS thus include functional and administrative independence of subsystems as well as geographic distribution [182]. Furthermore, the behavior of SoS is often emergent and its development evolutionary. This definition applies to numerous open systems, such as the power management case study considered in this thesis. Steghöfer [206] provides a detailed discussion of SoS as an organizational paradigm for open self-organizing systems.

Many MAS use predefined *static* hierarchies to reflect existing hierarchical structures. In autonomous power systems, such a hierarchy can resemble the organizational structure of utilities and grid operators, or the physical infrastructure of the power grid. Wedde [227], for instance, base the structure of a decentralized electricity market on the power grid's different voltage levels. The market's bottom-up principle allows the agents to balance power supply and demand as locally as possible. The hierarchy's internal nodes represent managers of balancing groups, whereas the leaves stand for physical producers and consumers. Prothmann [167] present a self-organizing traffic control system in which hierarchies consist of individual traffic lights, intersections, and entire roads. On each level, the system is able to learn traffic patterns used to adapt to the traffic flow, e.g., to create green waves during rush hour. The different levels allow the system to recognize patterns and to self-adapt on different scales. The information needed for such adaptations not only stems from the corresponding hierarchy level but is

also received from subordinate levels. As discussed at the beginning of this section, we deliberately abstain from aggregating information at higher levels of the hierarchy due to scalability issues. Instead, we use model abstraction to trade exact overall optimality for scalability.

The idea of handling complex problems in large-scale systems by means of hierarchical structures traces back to *hierarchical control systems* (cf. [139, 184]). In such systems, each component makes decisions according to the tasks and goals received from its superior and the feedback propagated up the hierarchy. The components representing the hierarchy's leaves perform the control actions as such. In the context of hierarchical control systems, *hierarchical task networks* (cf. [4, 44]) can be used to describe dependencies among tasks and subtasks, and allow for automated planning. In contrast to the self-organizing hierarchies regarded in this thesis, hierarchical control systems feature a fixed structure that is devised at design time.

Hierarchical problem decomposition – as a means to deal with the complexity of resource or task allocation in large-scale systems – has been successfully applied to various domains, including grid computing applications (cf. [2]), real-time systems (cf. [40]), smart grid applications (cf. [69]), and sensor webs (cf. [115]). A discussion of different approaches to hierarchical resource allocation can be found in [218].

As opposed to these related approaches, we focus on methods that allow for the creation of appropriate hierarchical structures at runtime. To comply with existing organizational structures or physical infrastructure, we show how a self-organizing hierarchy can evolve in partially predefined structures in Chapter 9.

Indeed, there are several approaches to the dynamic creation of hierarchies in the literature, mainly in the domains of sensor networks and grid computing. Wireless sensor networks typically consist of a large number of energy-constrained sensor nodes with limited transmission ranges that have to forward collected data to a processing element. In such a setting, hierarchical clustering algorithms are used to create energy-efficient overlay networks [28]. The internal nodes of such a hierarchy are sensor nodes acting in the role of cluster heads. Each cluster head serves as communication hub for a group of nearby sensor nodes. It receives information from subordinate nodes and forwards the aggregated data to its superordinate cluster head, ultimately reaching the (centralized) processing element. When forming the hierarchy, the number of hops (i.e., the nodes a message has to go through to reach its destination) and the nodes' communication range are decisive factors. In the distributed hierarchical clustering algorithm presented in [28], the sensor nodes create the hierarchy in a bottom-up and randomized manner. With regard to a specific hierarchy level, the cluster heads decide to become a cluster head for the next higher level with a certain probability. Those nodes that did not decide to become a cluster head join the closest of the new clusters. Of course, this procedure partitions the nodes of each hierarchy level. The combinatorial optimization problem the nodes encounter when forming the clusters, however, is trivial since they simply have to join the closest cluster. The algorithm is not designed for optimizing a partitioning with respect to multiple application-specific criteria. In applications in which the communication range is irrelevant, the algorithm therefore forms partitions in a completely randomized manner.

AETOS [166] is an approach to the self-organized creation of hierarchical overlay networks, e.g., for grid computing applications. In particular, Pournaras et al. [166] aim for robust hierarchical structures that are prepared for node failures. The authors assume that a malfunctioning node disconnects all subordinate nodes from the hierarchy. The goal is therefore to form a hierarchy in which a malfunctioning node disconnects as few nodes as possible. Because the number of disconnected nodes is likely to increase with the hierarchy level in which a failure occurs, Pournaras et al. [166] propose a formation algorithm that moves reliable nodes to higher positions in the hierarchy. While the hierarchy results from local interactions, AETOS only optimizes the vertical positioning of the nodes. The composition of partitions is not explicitly optimized. Because nodes prefer connections with reliable parents and children, the emerging partitions should contain nodes whose reliability is as high as possible but lower than the parent's reliability. Similarly, Wang et al. [224] introduce a self-organizing hierarchical overlay network for live video multicast. The structure is formed by adding nodes at random positions in the tree. To maximize the bandwidth and minimize the delay, the nodes swap positions to obtain a structure that reduces the number of hops while respecting the nodes' available bandwidth, a requirement that increases

with the hierarchy level. These and other approaches (e.g., [135, 209]) to self-organizing hierarchical overlay networks for grid computing applications, have in common that the formation problem is reduced to the selection of an appropriate parent within the hierarchy. The hierarchies' shape and quality depends on the employed parent selection strategy. In our approach, the agents self-organize into partitions according to application-specific optimization criteria. Having solved the PP, each created partition is assigned to a new or an existing parent (i.e., intermediary) representing the collective in the hierarchy (see Sections 6.4 and 9.1).

To summarize, the most important difference between our approach to self-organizing hierarchies and those from the body of literature is that our hierarchies evolve by recursively solving a partitioning problem in which the composition of the resulting partitions is optimized with regard to application-specific criteria. This difference primarily stems from the hierarchies' purpose. Our hierarchies aim at decomposing an optimization problem (viz., the scheduling problem) that has to be solved by the agents in cooperation. The agents decompose the problem by forming explicit agent organizations that are represented by intermediaries and arranged in a tree. Each organization (an AVPP in our case study) has to satisfy a part of the overall demand despite disturbances originating from participants that cannot or should not be excluded from the system. For this reason, the organizations have to exhibit a suitable *composition*. In Section 6.3, we explained that the agents should strive for homogeneous partitionings (i.e., similar organizations) to increase the system's stability and robustness. However, the system's scale prohibits a method that optimizes the entire hierarchical structure at once. This is why we opt for an approach to self-organizing hierarchies that recursively solves the PP defined in Section 6.1. Each instance of the PP focuses on optimizing the composition of organizations in a specific region of the hierarchy. To come to a compromise between solution quality and runtime performance when solving the scheduling problem, we propose to control the size and the number of created partitions by means of the partitioning constraints (see Section 6.2).

Kim and Candan [114] also follow the idea of creating a hierarchy by recursive partitioning but in the context of *graph partitioning*. Graph partitioning aims at partitioning an *existing* graph representing, e.g., a social network, into smaller subgraphs to be able to store or process the represented data more efficiently. Common graph partitioning approaches, such as *vertex-cut* or *edge-cut partitioning*, are, however, not applicable to our problem. While vertex-cut can assign a single node to multiple partitions and strives to minimize the number of such “ambiguous” assignments, edge-cut tries to minimize the number of edges between the resulting partitions (i.e., subgraphs). The graph and, in particular, its edges thus have to have clear semantics influencing the quality of a partitioning. In electric power systems, such an existing graph could represent the structure of a power grid. In future work, it would therefore be interesting to use graph partitioning to identify subnetworks in power grids. Within each of these subnetworks, our approach to self-organizing hierarchies could then be used to establish scalable, efficient, and stable system structures.

In the following, we outline problems that are closely related to the PP defined in this thesis, and review algorithms for their solution.

Problems Related to the Partitioning Problem

As stated in Section 6.1, the *set partitioning problem* (SPP) (cf. [27]) is closely related to the PP considered in this thesis. In the SPP, a set $\mathfrak{A} = \{a_1, \dots, a_n\}$ of $n > 1$ agents a_i (or objects, in a more general sense) is partitioned into non-empty and pairwise disjoint partitions that together constitute a partitioning \mathcal{P} at minimal cost. Feasible, i.e., valid, partitions $\mathfrak{B} = \{b_1, \dots, b_m\}$ (with $\forall b_j \in \mathfrak{B} : b_j \neq \emptyset \wedge b_j \subseteq \mathfrak{A}$) are assumed to be calculated before the actual optimization problem has to be solved. Furthermore, the costs c_j of having a partition b_j included in \mathcal{P} are additive and specified in advance. The SPP is usually

defined in the form of a 0-1 integer program (cf. [52]):

$$\underset{x_j}{\text{minimize}} \quad \sum_{j=1}^m c_j \cdot x_j \quad (6.7a)$$

$$\text{subject to} \quad \forall i \in \{1, \dots, n\} : \sum_{j=1}^m f_{ij} \cdot x_j = 1, \quad (6.7b)$$

$$\begin{aligned} \text{with} \quad & \forall j \in \{1, \dots, m\} : x_j \in \{0, 1\}, \\ & \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\} : f_{ij} \in \{0, 1\} \end{aligned}$$

In this definition, the vector \mathbf{x} stands for the partitioning \mathcal{P} . The vector component x_j equals 1 if \mathcal{P} should include the predefined partition b_j , and 0 otherwise. The f_{ij} define an $n \times m$ (0,1)-matrix that holds information about the composition of the predefined partitions. An entry f_{ij} equals 1 if agent a_i is contained in partition b_j , and 0 otherwise. The constraint in Equation (6.7b) ensures that each agent $a_i \in \mathfrak{A}$ is contained in exactly one partition in \mathcal{P} . It corresponds to Equations (6.2b) and (6.2c) in the definition of our PP. Equation (6.7a) represents the additive objective; the costs of \mathcal{P} are the sum of the predefined costs of the included partitions. Finding the optimal partitioning is NP-hard [72].

Balas and Padberg [27] survey problems related to the SPP, including the *set covering problem* and the *set packing problem*. In the set covering problem, the equality in Equation (6.7b) is replaced by “ \geq ” so that each agent has to be part of *at least one* partition in \mathcal{P} . The set packing problem replaces the equality by “ \leq ” so that each agent *must not be included in more than one* partition in \mathcal{P} , which corresponds to structures known from *strict partitioning clustering with outliers*. While the set covering problem does not satisfy the SPP’s requirement of pairwise disjoint partitions, the set packing problem does not satisfy the SPP’s requirement of creating a partitioning that covers the set of participating agents \mathfrak{A} . As explained in Section 6.3, the class of open systems regarded in this thesis prohibits the exclusion of agents from the system. Since our hierarchical system structure constitutes a tree, the partitions further have to be disjoint. The PP is therefore more similar to the SPP.

The SPP is sometimes extended by so-called *base constraints* that are of the form $d_{\min} \leq \sum_{j=1}^m d_j \cdot x_j \leq d_{\max}$ (with $0 < d_{\min} \leq d_{\max}$ and $\forall j \in \{1, \dots, m\} : d_j \geq 0$) [91]. The d_j can be regarded as another type of expenses that incur when partition b_j is contained in \mathcal{P} , and whose sum has to remain between the bounds d_{\min} and d_{\max} . If $\forall j \in \{1, \dots, m\} : d_j = 1$, the term $\sum_{j=1}^m d_j \cdot x_j$ counts the number of partitions in \mathcal{P} . In combination with $d_{\min} = n_{\min}$ and $d_{\max} = n_{\max}$, the base constraints can be used to restrict the number of partitions in \mathcal{P} to a suitable range as is the case with our partitioning constraints (see Equation (6.2e)). However, many algorithms solving the SPP are not designed for complying with base constraints (e.g., [52]) and are thus not able to abide by a given range for the number of partitions, which is needed to create suitable hierarchies. By contrast, all algorithms solving the original SPP can ensure that created partitionings adhere to the minimum- and maximum-partition-size constraints (see Equation (6.2d)). That is because, in the SPP, we can exclude all non-compliant partitions from the predefined set of feasible partitions.

A handful of variants of the SPP are discussed by Lamarche-Perrin et al. [129]. These impose different constraints on the set of feasible partitions \mathfrak{B} . In the *hierarchical SPP*, for instance, every two feasible partitions in \mathfrak{B} are either disjoint or one is a subset of the other so that $\forall b_i, b_j \in \mathfrak{B} : b_i \cap b_j = \emptyset \vee b_i \subseteq b_j \vee b_j \subseteq b_i$. Although it is thus possible to arrange the partitions \mathfrak{B} in a hierarchy, the resulting partitioning still represents a *flat* structure. In the *complete SPP*, all non-empty subsets of \mathfrak{A} are feasible partitions so that $\mathfrak{B} = \mathcal{P}(\mathfrak{A}) \setminus \emptyset$. The complete SPP is closely related to our PP if we do not restrict the size of partitions (i.e., $s_{\min} = 1$ and $s_{\max} = n$). The unbounded case often occurs in *coalition structure generation* [185].

The PP considered in this thesis differs from the SPP in four ways: (1) The PP presumes that feasible partitions are only constrained in terms of their size. (2) To allow the agents to guide the self-organized formation of the hierarchical system structure, the PP assumes the number-of-partitions constraints to be mandatory and used in addition to the partition-size constraints. (3) Preliminary evaluations confirmed our expectation that the large number of feasible partitions (this results from

point (1)), prevents us from calculating all of them in advance (see Section 6.1). This is especially the case if the set of agents \mathfrak{A} to be partitioned is subject to change over time. (4) The SPP supposes that the costs of having a feasible partition included in the partitioning are additive and predefined for all feasible partitions. However, this prohibits objective functions that assess the quality of a partitioning dependent on the relative composition of its partitions, as is the case with homogeneous partitioning. In the definition of our PP, we therefore only ask for an application-specific cost function that evaluates if a partitioning, i.e., a combination of partitions, is fit for purpose. Due to these differences, it is not possible to use algorithms developed for the original SPP to solve the PP in a general manner.

The PP can be regarded as a generalization of several problems that have been studied in various fields for a long time. In *coalition formation* (cf. [185]) – a key issue in MAS and game theory [180] –, the coalitions correspond to the partitions and the resulting coalition structure to the partitioning. Coalition formation involves solving the problem of *coalition structure generation*. The PP is equivalent to coalition structure generation if its objective function defines how well agents can work together on a common task. Similarly, we can form *teams* [30] by solving the PP. Approaches that do not call for disjoint coalitions create structures as defined by the set covering problem (cf. [199]). With respect to *cluster analysis*, the clusters represent the partitions and the clustering stands for the partitioning. If the objective function specifies to group similar objects, the PP is equivalent to *strict partitioning clustering (with outliers)* (cf. [138]). Otherwise if the clusters should be composed of dissimilar objects, the PP corresponds to *anticlustering* (cf. [203, 215]).

Algorithms Solving Variants of the Partitioning Problem

Algorithms solving the PP or its variants have a broad area of application, such as transportation systems (cf. [52]), sensor networks (cf. [232]), power systems (cf. [176]), flexible manufacturing systems (cf. [7]), network on chip (cf. [67]), e-commerce (cf. [45]), and image segmentation (cf. [160]). In the following, we give an overview of generic approaches to the original SPP, cluster analysis, and coalition formation on the one hand, and highly specialized algorithms solving specific variants of the PP on the other hand.

A well-known application of the original SPP in the area of transportation systems is the so-called *airline crew scheduling problem* [75]. Here, the goal is to find the least expensive set of feasible crew pairings (i.e., a feasible sequence of flight legs for a crew) that covers all scheduled flight legs exactly once. Chu and Beasley [52] introduce a *genetic algorithm* (GA) that solves the original SPP. Hence, their GA needs a pre-calculated set of feasible partitions. As discussed before, this is not possible in the setting regarded in this thesis. Their GA allows for the creation of infeasible interim solutions that are corrected by means of a heuristic. In theory, the GA could also be extended to respect prescribed ranges for the number of partitions by means of base constraints. However, it would not benefit from a reduced search space (as would be the case if it contained only feasible solutions), and it would be necessary to devise additional heuristics for correcting the interim solutions in order to obtain feasible (candidate) solutions at all. Another approach to the SPP respecting base constraints is outlined in [91]. The presented *branch-and-cut* algorithm exploits the SPP’s assumption of an additive objective function. Lamarche-Perrin et al. [129] introduce a generic algorithmic framework for designing algorithms solving special versions of the SPP. To obtain solutions in an efficient manner, the framework relies on *dynamic programming*. Dynamic programming solves an optimization problem by recursively breaking it down into simpler sub-problems, finding the optimal solution for these sub-problems, and combining these partial solutions to an optimal overall solution. This method requires that the problem to be solved has an *optimal substructure*, meaning that an optimal overall solution can be derived from optimal partial solutions (also known as Bellman’s “Principle of Optimality”). The SPP features an optimal substructure because of its additive objective function. Due to their assumptions, these approaches cannot be used to solve the PP.

In the context of cluster analysis, van der Merwe and Engelbrecht [217] propose a *particle swarm optimizer* (PSO) for data clustering. Each particle represents a complete solution to the clustering problem. Alam et al. [3] present an evolutionary PSO in which a new generation of particles can replace those particles contributing to a bad solution, which allows the algorithm to leave local optima.

Importantly, their particles represent partial instead of complete candidate solutions. In detail, each particle stands for a single cluster, comprising a centroid and its assigned elements. Both algorithms are specialized to forming clusters according to the k-means distance measure. As all approaches that just implement the *k-means* clustering algorithm [138], they are not able to deal with non-additive objective functions. Furthermore, the user has to specify the number of partitions k exactly. A suitable exact number of partitions is, however, often not known beforehand (further drawbacks of k-means, such as the formation of partitions of similar size, are discussed in [24]).

Other approaches, such as the *x-means* algorithm [98] or some *hierarchical clustering algorithms* [36], automatically find a suitable number of partitions for a given data set, but are not able to adhere to predefined ranges for the size and the number of partitions. A hierarchical clustering algorithm forms a tree of clusters in which clusters are refined from top to bottom. Such a tree can be created in a bottom-up (agglomerative) manner by recursively merging similar clusters, or in a top-down (divisive) manner by recursively splitting clusters. Popular examples of hierarchical clustering algorithms are *CURE* [79] and *BIRCH* [236]. While hierarchical clustering has many advantages over partitioning clustering algorithms (see [36] for a comparison), such as k-means or its relative *k-medoids* [106], the approach of optimizing partial solutions in separated branches is not properly suited for creating homogeneous partitionings, which requires to optimize the partitioning as a whole. In the domain of image segmentation, Omran et al. [160] present a dynamic clustering approach that identifies a suitable number of clusters for a given image in an iterative manner. For this purpose, the approach alternates between the application of a PSO and the k-means algorithm [138] to converge to a suitable number of clusters and an appropriate composition of the clusters, respectively. While the k-means algorithm could be replaced by other clustering algorithms, the approach is not able to comply with the partitioning constraints either.

Ogston et al. [159] present an agent- and graph-based clustering approach. The graph is used as an overlay network that constrains communication between agents on the one hand, and constitutes the current clustering on the other hand. Similarly to the k-means distance measure (see Equation (6.4)), the Euclidean distance is used to quantify the similarity between agents and clusters. Because the decision whether an agent should be included in or excluded from a cluster is only based on this distance, the approach is not able to deal with non-additive objective functions, such as homogeneous partitioning. Besides a maximum size of partitions, the algorithm is not able to comply with the partitioning constraints.

As outlined in Section 6.3, coalition formation addresses the issue of creating groups of agents that are able to fulfill one or more specific tasks in cooperation. To find the coalition structure of maximal utility, a characteristic function is used to assign a value to each possible coalition in accordance with the tasks to be performed. Usually, the value of a coalition structure is the sum of the values of the included coalitions [170, 199]. Several coalition formation algorithms (e.g., those based on *integer linear programs*) can yield optimal solutions, but require global system knowledge and can quickly run out of memory, even with a small number of agents to be partitioned [170]. Because of the PP's complexity, they are often designed as *anytime algorithms* [170]. To cope with large search spaces, Shehory and Kraus [199] distribute the entire search space among the agents to be able to pre-calculate the utility of all possible partitions and pick the best one after a global announcement. The number of feasible solutions is additionally reduced by restricting the maximum size of coalitions. Other approaches addressing the issue of large search spaces rely on dynamic programming or *branch-and-bound*, which, however, make specific assumptions about the employed objective function, e.g., when computing the lower bounds used for pruning [170]. Sandholm et al. [185] present an anytime algorithm for coalition structure generation whose results are guaranteed to be within a predefined bound from the optimum. The approach can deal with coalition values that are neither superadditive nor subadditive. As is the case with most coalition formation algorithms, the approach assumes that the size and the number of coalitions is not constrained, though. As mentioned before, some approaches to coalition formation only comply with the constraints of the set covering problem. Xu and Li [229] introduce a discrete PSO that can generate such overlapping coalitions.

In the field of network on chip, Faruque et al. [67] present a domain-specific agent-based clustering approach. Similarly to coalition formation, their formation process is driven by tasks, meaning that clusters are formed in such a way that their members can perform one or more specific tasks in cooperation.

Specifically, the algorithm is used to map tasks to processing elements (PEs). If a centralized task scheduler cannot map a task to a cluster, a re-clustering takes place. During this process, clusters try to integrate PEs that are idle or can be made available in time. Instead of short-lived task-driven partitionings, the class of open systems regarded in this thesis calls for stable partitionings that allow for robust resource allocation despite disturbances (see Section 6.3).

Rosinger and Beer [176] present a decentralized partitioning algorithm enabling small producers and consumers to participate in an autonomous electricity market. Producers and consumers self-organize into groups of trustworthy agents that together are able to provide active power products. The better an agent fulfilled previous contracts, the higher is its trust value. The partitioning algorithm is based on the *contract net protocol* [202]. Each partition is represented by a leader that tries to enlarge its partition until it is able to provide a certain product. For this purpose, the leader invites sufficiently trustworthy agents to join its partition. The invited agent only accepts the invitation if the current members of the requesting partition are sufficiently trustworthy. The protocol further allows the invited agent to propose a subset of its current partition members for integration. Again, the requesting leader accepts such a proposal if the proposed agents are sufficiently trustworthy. Unfortunately, the protocol ignores the underlying combinatorial optimization problem because of the following two inconsistencies: First, agents proposed by the invited agent are not able to check for the trustworthiness of the requesting partition. Second, a leader can invite several agents and accept several proposals at the same time. An invited agent does not know which other agents will join the partition, though. Apart from that, the approach is not applicable to the PP because partitions are only interested in improving their own composition without taking the other partitions' composition into account. Predefined ranges for the size and the number of partitions are not considered.

In sensor networks, clustering aims for energy-efficient overlay networks. Besides the hierarchical clustering algorithm by Bandyopadhyay and Coyle [28] that we discussed at the beginning of this section, there are several further approaches, such as ACE [49] or HEED [232]. Due to the inherent properties of sensor networks – such as the large number of distributed nodes –, these algorithms work in a completely decentralized manner. As explained before, the goal of these algorithms is to assign each sensor node a cluster head within its communication radius and to allow all cluster heads to communicate with each other. Because of their domain-specific assumptions, clustering algorithms for sensor networks are highly specialized and cannot be readily applied to other domains. While our self-organization algorithms presented in Chapters 7 and 8 allow for arbitrary objective functions, it would conversely be difficult to apply them in the context of sensor networks because they do not address the issues of energy efficiency or limited communication ranges.

In some cases, predefined organizational structures are exploited to guide the search for suitable partitionings. This enables the use of local knowledge and neighborhood relations. In the context of flexible manufacturing systems, Anders et al. [7] present a decentralized self-organization algorithm that is able to identify which part of a production line has to be reconfigured to compensate for a broken tool or robot. Such a part is represented by a partition whose formation requires a graph structure defining input/output relations between agents. Aiming for more efficient coalition formation, Abdallah and Lesser [1] describe an algorithm that is based on a hierarchical system structure. The concept of tasks and subtasks play a central role in the course of creating an adequate coalition structure. Rahimian et al. [169] provide a distributed balanced graph partitioning algorithm for edge-cut and vertex-cut partitioning. To find suitable partitionings, the algorithm uses local search and simulated annealing techniques and is performed on the basis of the graph to be partitioned. These approaches cannot be transferred to the PP, either due to their specialization or as they solve a completely different problem.

Summary

As outlined in this section, hierarchical system structures and hierarchical problem decomposition have been identified as a means to deal with complexity and scalability issues in various domains and applications. While some approaches assume static predefined structures, others are able to form hierarchies in a self-organizing manner at runtime. To be able to solve the scheduling problem in the class of open systems regarded in this thesis, the partitionings constituting the hierarchy have to be

optimized with regard to different application-specific formation criteria. To come to a compromise between solution quality and runtime performance, it must further be possible to control the size and the number of created partitions. For these reasons, we propose self-organizing hierarchies that evolve by recursively solving the PP. Existing approaches to self-organizing hierarchies do not meet these requirements. This is mainly because of their highly specialized nature (cf. approaches for sensor networks or grid computing).

Since finding the optimal partitioning constitutes an NP-hard combinatorial optimization problem, the PP calls for efficient solutions that can cope with large search spaces. For this reason, approaches solving variants of the PP propose to exploit properties of the regarded problem (such as optimal substructure or additivity) to represent the search space in a way that allows for a systematic search for high-quality solutions (cf. [129, 170]), to distribute the search space among the agents (cf. [199]), to reduce its size by excluding specific solutions in advance (cf. [138]), to trade optimality for efficiency by relying on metaheuristics – such as genetic algorithms (cf. [52]) or particle swarm optimizers (cf. [3]) –, or to handle complexity by solving the PP in a completely decentralized manner on the basis of local knowledge (cf. [159]). Because of the PP’s complexity, such approaches are often designed as anytime algorithms (cf. [170, 185]).

In a nutshell, algorithms for solving the PP are either (1) specialized to a particular problem in a certain domain (cf. the examples discussed above), (2) depend on the properties of a specific objective function, or are (3) very restrictive with regard to the possibilities for specifying mandatory characteristics of the resulting partitioning’s structure in the form of the number and the size of partitions. These attributes limit the algorithms’ applicability, especially with regard to the self-organized formation of hierarchies as a form of autonomous problem decomposition in large-scale MAS. As for point (2), many algorithms for the original SPP (e.g., [52]), coalition structure generation (e.g., [170]), or clustering (e.g., [138]) are specialized to certain objective functions. Most often, they assume that the quality of partitions is additive, which allows them to assess partitions independently of each other. While this also enables the application of specific optimization methods, such as dynamic programming, branch-and-bound, or branch-and-cut, these algorithms do not solve the PP in a general manner. With respect to point (3), most algorithms either do not allow the agents or the user to characterize valid partitionings at all (cf. [159]) or they have to be very specific (cf. [138] or [199]).

Our self-organization algorithms PSOPP and SPADA, which we introduce in Chapters 7 and 8, do not suffer from these drawbacks. While approaches optimized to specific problems in specific domains might make better use of domain-specific knowledge (cf. [232]), PSOPP and SPADA solve the PP in a general manner, independently of the characteristics of a specific objective function. By defining an application-specific objective function, they can be used for homogeneous partitioning, coalition structure generation, strict partitioning clustering (with outliers), anticlustering, as well as combinations of these different objectives. Moreover, PSOPP and SPADA are designed for the self-organized creation of hierarchical system structures (see Chapter 9). To this end, they provide solutions that comply with the partitioning constraints. PSOPP is a discrete particle swarm optimizer and thus builds on a population-based metaheuristic to cope with the PP’s complexity. SPADA, on the other hand, implements a decentralized agent-based approach. Furthermore, both approaches are anytime algorithms – a property that has proved to be beneficial when solving time-consuming optimization problems that might have to be interrupted [170].

Chapter Summary and Outlook

In this chapter, we introduced the core concepts of a self-organizing hierarchical system structure for large-scale open technical systems. We identified the partitioning problem as the underlying optimization problem that has to be solved by the agents to create and maintain a suitable hierarchy. Furthermore, we highlighted the necessity of specifying suitable ranges for the number and the size of partitions to guide the degree of problem decomposition. With the concept of homogeneous partitioning, we outlined a new organizational paradigm describing structures consisting of similar organizations that are equally able to cope with disturbances originating from a highly dynamic and uncertain environment. Our

evaluation confirmed that homogeneous partitioning leads to much more stable structures than dissimilar organizations consisting of similar agents.

In our discussion of the related work, we pointed out that existing algorithms cannot be used to create hierarchies satisfying the requirements imposed by the class of open systems regarded in this thesis. Consequently, there is a need for new self-organization algorithms that allow for the formation of adequate hierarchical structures on the basis of the above-mentioned concepts. In Chapters 7 and 8, we present two algorithms that solve the NP-hard partitioning problem in a general manner, i.e., independently of the characteristics of a specific objective function. Chapter 9 explains how the agents employ these algorithms to actually form a hierarchical structure.

PSOPP – a Particle Swarm Optimizer for Solving the Partitioning Problem

Summary. In the partitioning problem (PP), finding the optimal solution requires solving an NP-hard combinatorial optimization problem. For this reason, many approaches confine themselves to specific instances of the PP, e.g., those in which the objective function is additive. While this allows for the application of specialized heuristics (cf. the k-means algorithm) or algorithm design paradigms (cf. the branch-and-bound algorithm), such approaches do not solve the PP in a general manner. Another way of dealing with optimization problems whose complexity or large search space impede the search for high-quality solutions, are metaheuristics, such as particle swarm optimization or genetic algorithms. In this chapter, we present a discrete particle swarm optimizer that solves the NP-hard PP in a general manner. To be independent of the characteristics of a specific objective function, our algorithm relies on basic set operations to come to a solution. It is thus applicable to a broad range of applications. Among other things, it can be used for homogeneous partitioning, coalition structure generation, strict partitioning clustering, and anticlustering. Because our algorithm allows the agents to define valid partitioning constraints in terms of acceptable ranges for the number and the size of partitions, it can, combined with an additional control loop, even be used for the self-organized creation of hierarchical partitionings. Our evaluation confirms that it finds high-quality solutions in different scenarios and for various objectives in short time.

Publication. The concepts and results outlined in this chapter have been published in Anders et al. [15, 16].

As discussed in Section 6.5, one can find a plethora of metaheuristics solving problems related to the partitioning problem (PP) in the body of literature. This is mainly due to the problems' complexity. Besides genetic algorithms used to solve the original SPP (cf. [52, 76]), which requires a pre-calculated set of all feasible partitions (in Section 6.1, we argued that this is not feasible in large-scale MAS), the principle of *particle swarm optimization* (PSO) [107] has been applied to specific instances of the PP, such as data clustering (cf. [3, 217]). PSO is a biologically-inspired computational method and a population-based metaheuristic for optimization in large search spaces. In PSO, multiple particles, each representing a candidate solution, cooperatively explore the search space by exchanging information about promising positions in the search space.

In this chapter, we present *PSOPP*, a discrete *Particle Swarm Optimizer for the Partitioning Problem*. As opposed to related approaches (see Section 6.5), PSOPP (1) solves the PP in a general manner and (2) allows the system or the user to specify suitable ranges for the number as well as the size of partitions by means of the partitioning constraints introduced in Section 6.2. Because we define PSOPP's operations in a way that their application always maintains solution correctness, its particles comb through a search space that only contains feasible solutions. In our evaluation, we demonstrate

that excluding invalid partitionings from the search space is advantageous to PSOPP’s performance. Moreover, given that PSOPP is initialized with a correct candidate solution, it is an anytime algorithm. This means that it can provide a feasible solution at any time during the optimization process.

To be able to solve the general PP, we must not make any assumptions about the objective function assessing the quality of candidate solutions and steering the search for them. Therefore, our central idea – which could also be applied to other metaheuristics – is to rely on basic set operations to come to a solution. Since a partitioning is a set of sets of agents, set operations represent the lowest common denominator that can be used to optimize partitionings with regard to arbitrary objective functions. This concept differs from highly specialized heuristics, such as the k-means algorithm [138], or algorithm design paradigms, such as the branch-and-bound algorithm, that exploit specific characteristics of the objective function, such as additivity with regard to the costs of having a feasible partition included in the partitioning. To customize PSOPP to a specific application, it is therefore sufficient to devise an appropriate fitness function that evaluates the quality of candidate solutions. That way, PSOPP can be applied to many different applications in which solving the PP is relevant. It can be used for homogeneous partitioning, coalition structure generation, strict partitioning clustering (with outliers), anticlustering, as well as combinations of these heterogeneous objectives. In our power management case study, intermediaries use PSOPP for the formation of AVPPs. Recall that we have to take several different criteria on the basis of homogeneous partitioning, clustering, or anticlustering into account in order to form an adequate structure of AVPPs (see Section 6.3).

Since PSOPP’s solutions to the PP satisfy the partitioning constraints, it is predestined to create hierarchical system structures in a self-organized manner. To this end, it can be combined with an additional control loop presented in Chapter 9. In a hierarchical setting, the overall system is decomposed into a system of systems by recursively solving the PP. Since each subsystem is represented by an intermediary that encapsulates the essence of the agents it controls, we can often suppose that the intermediary has *regional* knowledge about its subordinates (that is, global knowledge with regard to its subsystem) [1]. As intermediaries do not need further information to create a suitable partitioning of their subordinates, PSOPP overcomes the drawbacks of strictly weak self-organization [195] in hierarchical systems.

By representing partitions and partitionings as *sets* (instead of, e.g., lists) and using set operations for their modification, PSOPP prevents an explosion of the size of the search space by avoiding *symmetries* (cf. [222]) in the combinatorial optimization problem that appear in a constraint model that encodes sets. In general, there are three different approaches to symmetry breaking in constraint programming [73]: Symmetries can be reduced or even removed completely by (1) reformulating the problem, (2) adding static symmetry breaking constraints before starting the search, or (3) dynamically adapting the search procedure at runtime. The latter can be achieved by adding static symmetry breaking constraints at runtime [222]. In contrast to constraint satisfaction problems, the challenge of dealing with symmetry in constraint optimization problems has not been paid much attention to and is thus a rather new focus of research [221]. Literature distinguishes between *variable symmetry* and *value symmetry* which is a bijection on the variables and their values, respectively, that preserves *feasibility*. Because the bijection does not necessarily guarantee that the value of the objective function is the same [221], further measures have to be taken to prevent that breaking symmetries impairs the quality of the solution. In PSOPP, we avoid variable symmetry while preserving the value of the objective function by using set data types as the underlying primitives.

When solving the PP by means of a metaheuristic, each agent $a_i \in \mathfrak{A}$ that is to be partitioned is represented by a vector \mathbf{g}_i of those attributes of a_i that are relevant to solve the PP. Therefore, we refer to the agents or the vectors to partition as “elements” in this chapter. If the agent’s unique identifier is irrelevant (i.e., not part of \mathbf{g}_i), we might have $\mathbf{g}_i = \mathbf{g}_j$ for the vectors of two agents $a_i \neq a_j$. In such a situation, we actually have to solve a *multiset partitioning problem* (MPP) for the multiset $\mathfrak{G} = \{\mathbf{g}_1, \dots, \mathbf{g}_n\}$. In the MPP, the multiset sum $\biguplus_{K \in \mathcal{P}} K$ of all partitions K (here, non-empty multisets of elements) in the partitioning \mathcal{P} must equal \mathfrak{G} . On the one hand, considering multisets instead of sets avoids further symmetries because we do not need to distinguish \mathbf{g}_i and \mathbf{g}_j by the identifiers of the corresponding agents. On the other hand, it also complicates the application of the operations used for optimization. In this chapter, we therefore assume that all vectors \mathbf{g}_i are different

(i.e., $\forall a_i, a_j \in \mathfrak{A} : a_i \neq a_j \Rightarrow \mathbf{g}_i \neq \mathbf{g}_j$). As indicated above, this assumption does not require that all agents are different because we can represent each agent $a_i \in \mathfrak{A}$ by a unique vector \mathbf{g}_i that contains a_i 's unique identifier as a mandatory component that is not regarded during the optimization process. As a result, \mathfrak{G} is a set so that the problem is reduced to a PP. The investigation whether the application of multiset operations actually leads to better solutions is subject to future work.

The remainder of this chapter is structured as follows: In Section 7.1, we give an introduction to the principle of PSO and some of its variants for combinatorial optimization. In Section 7.2, we present our algorithm, PSOPP. Based on a definition of similarity of partitionings outlined in Section 7.3, we explain the set operations PSOPP uses to comb through the search space in Sections 7.4 and 7.5. In Section 7.6, we discuss our evaluation results showing that PSOPP efficiently solves the PP in various scenarios.

7.1 Particle Swarm Optimization

PSO is a search heuristic for optimization problems. Its principle is based on the flocking behavior of birds or of schools of fish. Before we present a special form of PSO that is applicable to discrete optimization problems, such as the PP, we explain the basic idea of PSO.

General Definition

In the original definition of PSO [107], a swarm of *particles* moves around in an n -dimensional continuous search space in order to find near-optimal solutions. Moves are made in an iterative manner so that, in every iteration, each particle makes exactly one move. A particle's position in the search space represents a *candidate solution* to the optimization problem. Its quality is rated by a fitness function f : the higher the fitness, the better the solution. To be able to improve the quality of its candidate solution over time in a target-oriented manner, each particle Π_i is aware of *its best found solution* \mathcal{B}_i and the *best found solution* $\mathcal{B}_{\mathcal{N}_i}$ in its neighborhood \mathcal{N}_i ($\mathcal{B}_{\mathcal{N}_i} = \arg \max_{\mathcal{B} \in \{\mathcal{B}_j | \Pi_j \in \mathcal{N}_i\}} f(\mathcal{B})$). A particle's neighborhood is a sub-swarm of all particles and does not necessarily has to be static but might be adapted at runtime (cf. [133, 146]). Kennedy and Mendes [109] investigate the influence of different neighborhood topologies on the particles' performance. If a particle's neighborhood consists of all particles, $\mathcal{B}_{\mathcal{N}_i}$ corresponds to the *global best found solution* \mathcal{B} . The principle of PSO is based on the assumption that a promising candidate solution is likely to be close to or indicative of an even better solution.

Initially, particles usually start at random positions. In each iteration, the particles update their positions and best found solutions. The algorithm terminates, e.g., after a certain amount of iterations or if the particles converge to a (local) optimum. Its outcome is the global best found solution \mathcal{B} . In detail, a particle Π_i determines its *position* $\mathbf{x}_i(s+1)$ for the next iteration $s+1$ on the basis of its current position $\mathbf{x}_i(s)$ and its updated *velocity* $\mathbf{v}_i(s+1)$ as follows:

$$\mathbf{x}_i(s+1) = \mathbf{x}_i(s) + \mathbf{v}_i(s+1) \quad (7.1)$$

$$\mathbf{v}_i(s+1) = \omega \cdot \mathbf{v}_i(s) + c_1 \cdot r_1 \cdot (\mathcal{B}_i - \mathbf{x}_i(s)) + c_2 \cdot r_2 \cdot (\mathcal{B}_{\mathcal{N}_i} - \mathbf{x}_i(s)) \quad (7.2)$$

$$\text{with } \omega, c_1, c_2 \in \mathbb{R}_0^+, r_1, r_2 \in [0, 1], \text{ and } \forall s : \mathbf{x}_i(s), \mathbf{v}_i(s), \mathcal{B}_i, \mathcal{B}_{\mathcal{N}_i} \in \mathbb{R}^n$$

Since $\mathbf{v}_i(s+1)$ depends on the current velocity $\mathbf{v}_i(s)$, it embodies a certain *inertia* for the purpose of exploration. To search in promising regions of the search space, a particle's motion is further influenced by its best found solution \mathcal{B}_i and the best found solution $\mathcal{B}_{\mathcal{N}_i}$ in its neighborhood. As there is always a trade-off between exploration and exploitation, the constants ω , c_1 , and c_2 allow for establishing an appropriate balance between the particle's inertia and its attraction towards \mathcal{B}_i and $\mathcal{B}_{\mathcal{N}_i}$. The random numbers r_1 and r_2 are regenerated in every iteration. These interrelations are illustrated for a single particle in Figure 7.1a.

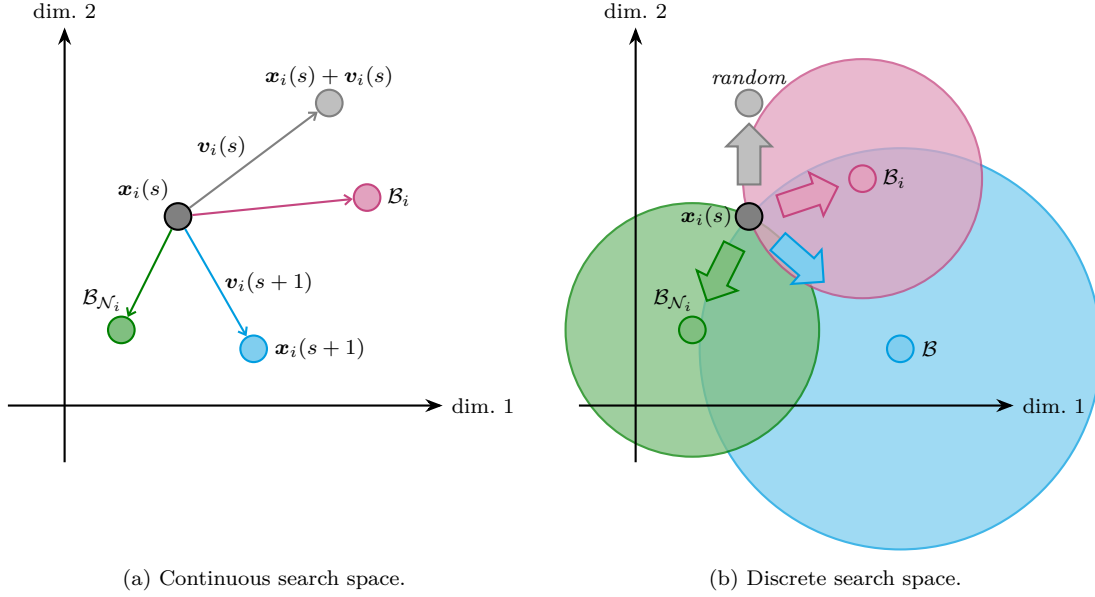


Figure 7.1: Motion of a single particle in a two-dimensional search space: In the original definition of PSO, the particle moves through a continuous search space (see Figure 7.1a). Its new position $\mathbf{x}_i(s+1)$ is determined by its updated velocity $\mathbf{v}_i(s+1)$, which is a linear combination of the particle's current velocity $\mathbf{v}_i(s)$, the vector from its current position $\mathbf{x}_i(s)$ to its best found solution \mathcal{B}_i , and the vector from $\mathbf{x}_i(s)$ to its best found solution \mathcal{B}_{N_i} in its neighborhood. Regarding JPSO, the particle moves through a discrete search space (see Figure 7.1b). Its new position results from either making a random move or from approaching \mathcal{B}_i , \mathcal{B}_{N_i} , or the global best found solution \mathcal{B} . In case of an approach, the particle's similarity to \mathcal{B}_i , \mathcal{B}_{N_i} , or \mathcal{B} increases so that its new position is somewhere within the corresponding circle depicted in Figure 7.1b. Note that $\mathbf{x}_i(s)$ lies on the circles' circumference.

Discrete Particle Swarm Optimization

The original definition of PSO is not applicable to discrete, e.g., combinatorial, optimization problems, such as the PP. Kennedy and Eberhart [108] extend the applicability of PSO to n -dimensional binary search spaces by introducing *Discrete PSO* (DPSO) in which the positions $\mathbf{x}_i(s)$, \mathcal{B}_i , and \mathcal{B}_{N_i} are values of the domain $\{0,1\}^n$. While the domain and the definition of the velocity $\mathbf{v}_i(s+1) \in \mathbb{R}^n$ as given in Equation (7.2) are not modified, the semantics of the velocity changes. In contrast to the original definition, each component $(v_i(s+1))_j \in \mathbb{R}$ of the vector $\mathbf{v}_i(s+1)$ represents a probability that the j -th component of the particle's position $\mathbf{x}_i(s+1)$ is either 0 or 1. Equation (7.1) therefore becomes invalid.

Another DPSO approach, which is called *Jumping PSO* (JPSO) [70], omits the concept of the velocity as defined in [107]. In simplified terms, JPSO redefines the motion of particles by replacing the linear combinations in Equation (7.1) and Equation (7.2) by an “either-or” operation that makes them “jump” through a multidimensional discrete search space \mathbb{S} :

$$\mathbf{x}_i(s+1) = \begin{cases} rdm(\mathbf{x}_i(s)) & \text{if } r_i \leq c_{rdm} \\ appr(\mathbf{x}_i(s), \mathcal{B}_i) & \text{if } c_{rdm} < r_i \leq c_{\mathcal{B}_i}^* \\ appr(\mathbf{x}_i(s), \mathcal{B}_{N_i}) & \text{if } c_{\mathcal{B}_i}^* < r_i \leq c_{\mathcal{B}_{N_i}}^* \\ appr(\mathbf{x}_i(s), \mathcal{B}) & \text{else} \end{cases} \quad (7.3)$$

with $r_i \in [0, 1]$, $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}_{N_i}}, c_{\mathcal{B}} \in [0, 1]$, $c_{rdm} + c_{\mathcal{B}_i} + c_{\mathcal{B}_{N_i}} + c_{\mathcal{B}} = 1$,

$c_{\mathcal{B}_i}^* = c_{rdm} + c_{\mathcal{B}_i}$, $c_{\mathcal{B}_{N_i}}^* = c_{rdm} + c_{\mathcal{B}_i} + c_{\mathcal{B}_{N_i}}$, and $\forall s : \mathbf{x}_i(s), \mathcal{B}_i, \mathcal{B}_{N_i}, \mathcal{B} \in \mathbb{S}$

Equation (7.3) states that a particle Π_i either makes a random move $rdm(\mathbf{x}_i(s))$ with a probability of

c_{rdm} or approaches $appr(\mathbf{x}_i(s), \beta)$ a specific candidate solution $\beta \in \{\mathcal{B}_i, \mathcal{B}_{\mathcal{N}_i}, \mathcal{B}\}$ with a probability of $c_{\mathcal{B}_i}$, $c_{\mathcal{B}_{\mathcal{N}_i}}$, or $c_{\mathcal{B}}$, respectively. In each iteration, this direction is determined by a random number r_i that is generated individually for each particle. Similarly to Equation (7.2), the constants c_{rdm} , $c_{\mathcal{B}_i}$, $c_{\mathcal{B}_{\mathcal{N}_i}}$, and $c_{\mathcal{B}}$ stipulate the particles' attitude towards exploration and exploitation. The idea of JPSO has been successfully applied to a number of high-dimensional combinatorial problems (see, e.g., [54, 194]).

Figure 7.1b summarizes the behavior of particles in JPSO. For approaching another candidate solution, JPSO requires a notion of *distance* or *similarity* between two candidate solutions that fits the regarded optimization problem. We present such a similarity metric for the PP in Section 7.3.

7.2 Basic Procedure of PSOPP

As for our algorithm, *PSOPP*, each particle embodies a solution to the PP, i.e., a partitioning of the set of elements \mathfrak{G} . PSOPP is inspired by DPSO's derivative JPSO outlined in the previous section. The motion of particles is thus not subject to inertia, i.e., a particle's position $\mathbf{x}_i(s+1)$ for the next iteration $s+1$ does not depend on the modifications made to move from its previous position $\mathbf{x}_i(s-1)$ to its current position $\mathbf{x}_i(s)$. In PSOPP, a particle's motion is influenced by its best found solution \mathcal{B}_i and the best found solution $\mathcal{B}_{\mathcal{N}_i}$ in its neighborhood \mathcal{N}_i . This complies with the general definition of PSO (cf. Equation (7.2)). While we could easily extend PSOPP such that its particles' motion is additionally influenced by the global best solution \mathcal{B} – as is the case with JPSO –, not including \mathcal{B} reduces the chance of getting trapped in local optima. With respect to the definition of JPSO's behavior in Equation (7.3), this corresponds to a probability of $c_{\mathcal{B}} = 0$.

As stated before, PSOPP allows for specifying mandatory characteristics of a feasible solution, i.e., partitioning, by means of the partitioning constraints defined in Section 6.2. These prescribe the minimum n_{min} and the maximum n_{max} number of partitions ($1 \leq n_{min} \leq n_{max} \leq |\mathfrak{G}|$) as well as their minimum s_{min} and maximum s_{max} size ($1 \leq s_{min} \leq s_{max} \leq |\mathfrak{G}|$). Because we define PSOPP's operations for the particles' motion in a way that always preserves the correctness of candidate solutions with respect to the partitioning constraints, partitionings that do not meet them are not represented in the search space. In other words, candidate solutions are restricted to the solution space. As we show in our evaluation in Section 7.6, suitable boundaries can thus lower the time needed to find high-quality solutions.

Having defined valid partitionings by means of n_{min} , n_{max} , s_{min} , s_{max} as well as the particles' attitude towards exploration and exploitation by fixing the constants c_{rdm} , $c_{\mathcal{B}_i}$, $c_{\mathcal{B}_{\mathcal{N}_i}}$, PSOPP creates a predefined number of particles at random or predetermined positions. The latter is especially suitable when a reorganization of an existing system structure has to take place: If the current structure does not contradict the partitioning constraints, it can be used as a starting point for the self-organization process (in Section 6.4, we showed how an invalid partitioning can be corrected so that it complies with the partitioning constraints). Mixing predefined and randomly generated initial partitionings encourages diversity. When searching for an initial system structure, particles are created at random positions. The number of particles does not change during PSOPP's runtime.

The position $\mathbf{x}_i(s)$ of each particle Π_i represents a partitioning \mathcal{P} (hereinafter, we use \mathcal{P} synonymously for $\mathbf{x}_i(s)$) that consists of $n_{min} \leq |\mathcal{P}| \leq n_{max}$ partitions. Every partition $K \in \mathcal{P}$ comprises $s_{min} \leq |K| \leq s_{max}$ elements. All particles concurrently explore the search space in search of better solutions by iteratively modifying their current positions (at random or by approaching other solutions) as long as a specific termination criterion is not met. In each iteration, a particle Π_i performs the following actions that are also depicted in Figure 7.2:

1. Evaluate the fitness $f(\mathcal{P})$ of the represented partitioning \mathcal{P} .
2. If the particle's fitness $f(\mathcal{P})$ is higher than the fitness $f(\mathcal{B}_i)$ of its best found solution \mathcal{B}_i , set \mathcal{B}_i to \mathcal{P} . Further, inform all other particles Π_j that contain Π_i in their neighborhood \mathcal{N}_j about the improvement so that they can update $\mathcal{B}_{\mathcal{N}_j}$, i.e., the best found solution in their neighborhood.
3. Update the best found solution $\mathcal{B}_{\mathcal{N}_i}$ in the particle's neighborhood \mathcal{N}_i on the basis of the improvements achieved by the particle's neighbors.

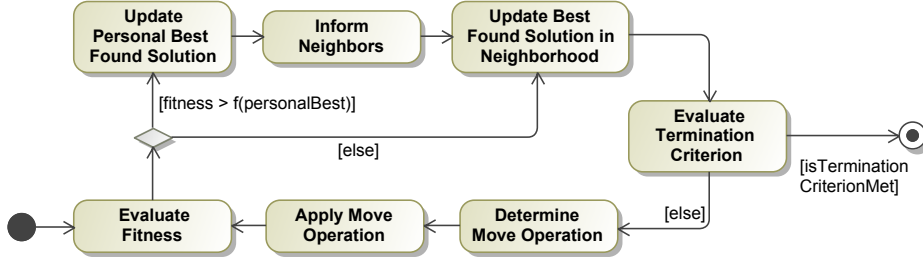


Figure 7.2: Actions performed by a particle in each iteration.

4. Stop if the termination criterion is met.
5. Otherwise, opt for the direction in which to move by generating the random number $r_i \in [0, 1]$ on the basis of a uniform distribution, i.e., choose whether a random move or an approach operation should be applied. In case of an approach operation, r_i also determines whether \mathcal{B}_i or $\mathcal{B}_{\mathcal{N}_i}$ should be approached. This behavior is described in Equation (7.3).
6. Determine the new position \mathcal{P}' by applying the selected move operation to \mathcal{P} .

Once all particles terminated, PSOPP returns the best found solution \mathcal{B} . Possible termination criteria are, e.g., a predefined amount of time, a predefined number of iterations (i.e., moves through the search space), a predefined threshold for the minimum fitness value, or a combination of these criteria. As the condition for termination can be formulated as a conjunction of hard constraints, PSOPP's termination can be coupled with the corridor of correct behavior (see Section 1.4). To obtain a meaningful system behavior, the constraints have to be consistent with those used to trigger reorganizations. With regard to the power management case study, we limit the runtime of a reorganization because the AVPPs must have sufficient time to recalculate schedules before their expiration date is exceeded.

Because PSOPP's particles are initialized with feasible candidate solutions (see Section 6.4) and its operations abide by the partitioning constraints, its candidate solutions are restricted to the solution space. For this reason, PSOPP is sound. Its completeness depends on the employed termination criterion, though: Just like every heuristic, PSOPP trades completeness for efficiency if a certain set of solutions has to be found, as is the case with the minimum-fitness-value criterion. On the other hand, PSOPP is complete if the termination criterion does not demand for specific characteristics of a solution that go beyond the scope of the partitioning constraints. In case of a predefined time limit, for instance, it can return an arbitrary candidate solution.

7.3 Similarity of Partitionings

The purpose of an approach operation is to increase the *similarity* (that is, to decrease the distance) between two partitionings \mathcal{P} and \mathcal{Q} by assimilating characteristics from \mathcal{Q} into \mathcal{P} . With regard to the search space, the idea of approaching \mathcal{Q} is that the particle Π_i representing \mathcal{P} might find better solutions in the neighborhood of \mathcal{Q} . Therefore, \mathcal{Q} stands for either \mathcal{B}_i or $\mathcal{B}_{\mathcal{N}_i}$. In this section, we define the *similarity* of partitionings on the basis of a definition by Kudo and Murai [126]. Note that the similarity does not give an indication of how many operations/moves are necessary to transfer one partitioning into another (i.e., to move from one position to another). Instead, it compares partitionings with regard to their composition. According to [126], the similarity of two partitionings \mathcal{P}, \mathcal{Q} is based on the definitions of a *refinement* and the *intersection* of two partitionings.

Definition 3 (Refinement). Partitioning \mathcal{P} is a **refinement** $\text{ref}(\mathcal{P}, \mathcal{Q})$ of partitioning \mathcal{Q} if and only if all partitions $K \in \mathcal{P}$ are subsets of partitions $L \in \mathcal{Q}$:

$$\text{ref}(\mathcal{P}, \mathcal{Q}) := \Leftrightarrow \forall K \in \mathcal{P} : \exists L \in \mathcal{Q} : K \subseteq L \quad (7.4)$$

Hence, if \mathcal{P} is a refinement of \mathcal{Q} , \mathcal{P} does not contain less partitions than \mathcal{Q} (i.e., $|\mathcal{P}| \geq |\mathcal{Q}|$). For instance, $\mathcal{P}_* = \{\{g_1, g_2\}, \{g_3\}, \{g_4\}\}$ is a refinement of $\mathcal{Q}_* = \{\{g_1, g_2, g_3\}, \{g_4\}\}$, whereas $\mathcal{R}_* = \{\{g_1, g_2, g_4\}, \{g_3\}\}$ is not because $\{g_1, g_2, g_4\}$ is not a subset of any partition in \mathcal{Q}_* . Note that we use the indices “*” and “+” to highlight examples of partitions or partitionings in this chapter.

Definition 4 (Intersection of Partitionings). *The **intersection** $\mathcal{P} \cap \mathcal{Q}$ of two partitionings \mathcal{P}, \mathcal{Q} is the set of all non-empty intersections of partitions in \mathcal{P} and \mathcal{Q} :*

$$\mathcal{P} \cap \mathcal{Q} = \{K \cap L \mid K \in \mathcal{P} \wedge L \in \mathcal{Q} \wedge K \cap L \neq \emptyset\}$$

Note that an intersection $\mathcal{P} \cap \mathcal{Q}$ is always a refinement of \mathcal{P} and \mathcal{Q} . For example, the intersection $\mathcal{S}_* \cap \mathcal{Q}_* = \{\{g_1, g_2\}, \{g_3\}, \{g_4\}\}$, which equals \mathcal{P}_* in the example above, is a refinement of $\mathcal{S}_* = \{\{g_1, g_2\}, \{g_3, g_4\}\}$ and $\mathcal{Q}_* = \{\{g_1, g_2, g_3\}, \{g_4\}\}$. Furthermore, if \mathcal{P} is a refinement of a partition \mathcal{Q} , we have $\mathcal{P} \cap \mathcal{Q} = \mathcal{P}$ for their intersection because every partition in \mathcal{P} is a subset of a partition in \mathcal{Q} .

Definition 5 (Similarity of Partitionings). *The **similarity** $\text{sim}(\mathcal{P}, \mathcal{Q}) \in]0, 1]$ of two non-empty partitionings \mathcal{P}, \mathcal{Q} is directly proportional to the ratio of the sum of their cardinalities to the cardinality of their intersection:*

$$\text{sim}(\mathcal{P}, \mathcal{Q}) := \frac{|\mathcal{P}| + |\mathcal{Q}|}{2 \cdot |\mathcal{P} \cap \mathcal{Q}|} \quad (7.5)$$

According to this definition, the more elements are in the same partitions in \mathcal{P} and \mathcal{Q} (i.e., the more elements constitute the *partitions’* intersection), the smaller the cardinality of the intersection $\mathcal{P} \cap \mathcal{Q}$ is and thus the more similar are the partitionings. In other words, the intersection $\mathcal{P} \cap \mathcal{Q}$ (which is a refinement of \mathcal{P} and \mathcal{Q}) should be as similar as possible to \mathcal{P} and \mathcal{Q} . Hence, $\text{sim}(\mathcal{P}, \mathcal{Q}) = 1$ if and only if $\mathcal{P} = \mathcal{Q}$, because then $\mathcal{P} \cap \mathcal{Q} = \mathcal{P} = \mathcal{Q}$. While not required, this definition of similarity does not allow us to compare two similarity values $\text{sim}(\mathcal{P}, \mathcal{Q})$ and $\text{sim}(\mathcal{R}, \mathcal{S})$ if they stem from four different partitionings. Regarding the two examples above,

$$\text{sim}(\mathcal{S}_*, \mathcal{Q}_*) = \frac{|\{\{g_1, g_2\}, \{g_3, g_4\}\}| + |\{\{g_1, g_2, g_3\}, \{g_4\}\}|}{2 \cdot |\{\{g_1, g_2\}, \{g_3\}, \{g_4\}\}|} = \frac{2+2}{2 \cdot 3} = \frac{4}{6}$$

is smaller than

$$\text{sim}(\mathcal{P}_*, \mathcal{Q}_*) = \frac{|\{\{g_1, g_2\}, \{g_3\}, \{g_4\}\}| + |\{\{g_1, g_2, g_3\}, \{g_4\}\}|}{2 \cdot |\{\{g_1, g_2\}, \{g_3\}, \{g_4\}\}|} = \frac{3+2}{2 \cdot 3} = \frac{5}{6}$$

since \mathcal{P}_* is a refinement of \mathcal{Q}_* . So assuming a particle at position \mathcal{S}_* , it can approach position \mathcal{Q}_* by splitting the partition $\{g_3, g_4\}$ into two partitions $\{g_3\}, \{g_4\}$. This modification transforms \mathcal{S}_* into \mathcal{P}_* .

Based on these definitions, we show that the operations enabling particles to approach each other always increase the similarity of the represented partitionings (see Section 7.5). Before we explain these operations in detail, we introduce the basic operations by means of random moves in the search space.

7.4 Random Moves in the Search Space

The motion of particles is a key factor in PSO because it is the only means to find better candidate solutions. As a solution to the PP is a partitioning (that is a set of sets), the motion of particles in the search space can be realized by the two set operations *split* and *join* [19]. In each iteration, each PSOPP particle makes exactly one move, either in a random direction or by approaching a specific position in the search space in a target-oriented manner (see Section 7.5). The corresponding operator is randomly selected. In this section, we concentrate on *random moves*, i.e., operators that modify the represented partitioning at random. In case the selected operator cannot be applied without violating a constraint, another operator is chosen. Because of the partitioning constraints and the restriction of candidate solutions to the solution space, there are situations in which neither the split nor the join operator can be applied. For such situations, we introduce an additional *exchange* operation.

Example – Running Example of Random Moves

Unless otherwise stated, we use $\mathcal{P}_* = \{\{g_1, g_2, g_4, g_5\}, \{g_3, g_6\}, \{g_7, g_8\}\}$ with partitions $K_* = \{g_1, g_2, g_4, g_5\}$, $L_* = \{g_3, g_6\}$, and $M_* = \{g_7, g_8\}$, $s_{min} = n_{min} = 2$, and $s_{max} = n_{max} = 4$ to illustrate the operators' application.

Random Split

The split operation divides a *randomly splittable partition* $K \in \mathcal{P}$ into two new non-empty disjoint partitions L, M such that $K = L \cup M$. For the resulting partitioning \mathcal{P}' , we have $\mathcal{P}' = (\mathcal{P} \setminus \{K\}) \cup \{L, M\}$. Because the resulting partitions L and M both have to fulfill the minimum-size constraint (i.e., $|L|, |M| \geq s_{min}$), the split operation can only be applied if partition K is big enough, i.e., if $|K| \geq 2 \cdot s_{min}$. Compared to the original partitioning \mathcal{P} , the split operation increases the number of partitions $|\mathcal{P}'|$ of the resulting partitioning \mathcal{P}' by one (i.e., $|\mathcal{P}'| = |\mathcal{P}| + 1$). To ensure that \mathcal{P}' also complies with the maximum number of partitions n_{max} , the split operation can only be applied if \mathcal{P} contains less than n_{max} partitions. Summarizing, \mathcal{P} 's set of *randomly splittable partitions* $\sigma_{rdm}(\mathcal{P})$ is defined as:

$$\sigma_{rdm}(\mathcal{P}) = \{K \mid K \in \mathcal{P} \wedge |K| \geq 2 \cdot s_{min} \wedge |\mathcal{P}| < n_{max}\}$$

Example – Random Split

In our example \mathcal{P}_* , only $K_* = \{g_1, g_2, g_4, g_5\}$ is randomly splittable, resulting, e.g., in a partitioning $\mathcal{P}'_* = \{\{g_1, g_2\}, \{g_4, g_5\}, \{g_3, g_6\}, \{g_7, g_8\}\}$.

Random Join

The join operation merges a *randomly joinable partition* $K \in \mathcal{P}$ and a *randomly joinable counterpart* $L \in \mathcal{P}$ (with $K \neq L$) into a single new partition $M = K \cup L$. For the resulting partitioning \mathcal{P}' , we have $\mathcal{P}' = (\mathcal{P} \setminus \{K, L\}) \cup \{K \cup L\}$. Because M has to satisfy the maximum-size constraint, L must be a partition that can be merged with K without exceeding the maximum allowed size, i.e., $|K| + |L| \leq s_{max}$. Since the join operator decreases the number of partitions in the resulting partitioning \mathcal{P}' by one, the operator can only be applied if \mathcal{P} features a sufficient number of partitions, i.e., if $|\mathcal{P}| > n_{min}$. Otherwise, \mathcal{P}' would violate the minimum-number-of-partitions constraint. Summarizing, \mathcal{P} 's sets of *randomly joinable partitions* $\iota_{rdm}(\mathcal{P})$ and *randomly joinable counterparts* $\iota_{rdm}^{\leftrightarrow}(K, \mathcal{P})$ are defined as follows:

$$\begin{aligned} \iota_{rdm}(\mathcal{P}) &= \{K \mid K \in \mathcal{P} \wedge \iota_{rdm}^{\leftrightarrow}(K, \mathcal{P}) \neq \emptyset \wedge |\mathcal{P}| > n_{min}\} \\ \iota_{rdm}^{\leftrightarrow}(K, \mathcal{P}) &= \{L \mid L \in \mathcal{P} \wedge |K| + |L| \leq s_{max} \wedge K \neq L\} \end{aligned}$$

Example – Random Join

With regard to our example \mathcal{P}_* , randomly joinable partitions are $L_* = \{g_3, g_6\}$ and $M_* = \{g_7, g_8\}$ with randomly joinable counterparts $\{M_*\}$ and $\{L_*\}$, respectively. Merging L_* and M_* yields $\mathcal{P}'_* = \{\{g_1, g_2, g_4, g_5\}, \{g_3, g_6, g_7, g_8\}\}$.

Random Exchange

Obviously, there are situations in which neither the split nor the join operator can be applied (particles must not violate the constraints temporarily since the search space only contains feasible solutions). For example, if $s_{min} = s_{max}$ or $n_{min} = n_{max}$, not a single particle is able to make a move using the split or the join operation. But even if $s_{min} \neq s_{max}$ and $n_{min} \neq n_{max}$, specific combinations of $s_{min}, s_{max}, n_{min}$, and n_{max} can cause individual particles to freeze: For instance, consider a partitioning

$\mathcal{P}_+ = \{\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}, \{\mathbf{g}_4, \mathbf{g}_5, \mathbf{g}_6\}\}$ with $s_{\min} = 2$, $s_{\max} = 4$, $n_{\min} = 2$, and $n_{\max} = 3$. Splitting one of the two partitions is not possible because of $s_{\min} = 2$ and the join operation is not applicable because of $n_{\min} = 2$. To prevent the particles from becoming jammed, we additionally introduce an *exchange* operator that atomically swaps some of the elements of two partitions.

The exchange operation interchanges the proper subset $\hat{K} \subset K$ (with $\hat{K} \neq \emptyset$) and the subset $\hat{L} \subseteq L$ (\hat{L} is allowed to be the empty set \emptyset) between a *randomly exchangeable partition* $K \in \mathcal{P}$ and a *randomly exchangeable counterpart* $L \in \mathcal{P}$. Using the non-empty proper subset \hat{K} of K avoids that the operation has no effect at all (as would be the case if all or no elements of K were integrated into L and vice versa). We deliberately allow \hat{L} to be empty in order to handle situations as given in the example above: Regarding partitioning $\mathcal{P}_+ = \{\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}, \{\mathbf{g}_4, \mathbf{g}_5, \mathbf{g}_6\}\}$, we can simply move $\hat{K}_+ = \{\mathbf{g}_3\}$ from $K_+ = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3\}$ into $L_+ = \{\mathbf{g}_4, \mathbf{g}_5, \mathbf{g}_6\}$. If we did not allow $\hat{L} = \emptyset$, we would have to perform two consecutive exchange operations to achieve the same result: In the example, we could, e.g., exchange $\{\mathbf{g}_2, \mathbf{g}_3\}$ and $\{\mathbf{g}_4\}$, and finally $\{\mathbf{g}_4\}$ and $\{\mathbf{g}_2\}$.

Basically, the exchange operation corresponds to a join that is followed by a split. Since \hat{K} is a non-empty proper subset of K , as the split operation, it yields two non-empty partitions. Because an exchange between two partitions of size one would either not have any effect or contradict this characteristic (and thus correspond to a join), we define \mathcal{P} 's sets of *randomly exchangeable partitions* $\epsilon_{rdm}(\mathcal{P})$ and *randomly exchangeable counterparts* $\epsilon_{rdm}^{\leftrightarrow}(K, \mathcal{P})$ as:

$$\begin{aligned}\epsilon_{rdm}(\mathcal{P}) &= \{K \mid K \in \mathcal{P} \wedge |K| > 1\} \\ \epsilon_{rdm}^{\leftrightarrow}(K, \mathcal{P}) &= \mathcal{P} \setminus \{K\}\end{aligned}$$

When integrating an arbitrary non-empty proper subset $\hat{K} \subset K$ into L , \hat{K} as well as the subset $\hat{L} \subseteq L$ that is integrated into K must be specified in a way that the condition $|K'|, |L'| \in [s_{\min}, s_{\max}]$ holds for the resulting partitions K', L' . To achieve this, PSOPP's choice concerning \hat{L} is subject to the cardinality of \hat{K} . While $|\hat{K}|$ must be between 1 and $|K| - 1$ to ensure that \hat{K} is a non-empty proper subset of K , the following two constraints must hold for the randomly determined set \hat{L} to guarantee that the resulting partitioning $\mathcal{P}' = (\mathcal{P} \setminus \{K, L\}) \cup \{(K \setminus \hat{K}) \cup \hat{L}, (L \setminus \hat{L}) \cup \hat{K}\}$ respects s_{\min} and s_{\max} :

$$|\hat{L}| \leq \min \left\{ |L|, \min \left\{ (|L| + |\hat{K}|) - s_{\min}, s_{\max} - (|K| - |\hat{K}|) \right\} \right\} \quad (7.6)$$

$$|\hat{L}| \geq \max \left\{ \max \left\{ 0, s_{\min} - (|K| - |\hat{K}|) \right\}, (|L| + |\hat{K}|) - s_{\max} \right\} \quad (7.7)$$

Note that Equation (7.6) addresses $|L'| \geq s_{\min}$ and $|K'| \leq s_{\max}$, whereas Equation (7.7) ensures that $|K'| \geq s_{\min}$ and $|L'| \leq s_{\max}$.

Example – Random Exchange

In our example \mathcal{P}_* , we can, e.g., exchange $\hat{K}_* = \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_5\}$ and $\hat{M}_* = \{\mathbf{g}_7\}$ between K_* and M_* , resulting in $\mathcal{P}'_* = \{\{\mathbf{g}_4, \mathbf{g}_7\}, \{\mathbf{g}_3, \mathbf{g}_6\}, \{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_5, \mathbf{g}_8\}\}$.

There are two situations that obstruct the application of the random exchange operator: First, if all partitions are singletons (i.e., if $|\mathcal{P}| = |\mathfrak{G}|$), there is no randomly exchangeable partition, i.e., $\epsilon_{rdm}(\mathcal{P}) = \emptyset$. In such a case, a particle can use the random join operator to change its position if $n_{\min} < |\mathfrak{G}|$ and $s_{\max} \geq 2$. Second, if there is only a single partition, the set of randomly exchangeable counterparts $\epsilon_{rdm}^{\leftrightarrow}(K, \mathcal{P})$ is empty. Here, the random split operator can be used if $|\mathfrak{G}| \geq 2 \cdot s_{\min}$ and $n_{\max} \geq 2$.

The three operations split, join, and exchange allow PSOPP to create new or dissolve existing partitions or to swap elements between them while maintaining the properties of a partitioning and complying with the partitioning constraints. In this section, we focused on random moves, where we cannot make any statement with regard to the change in similarity to another partitioning. In the next section, we explain how particles use the basic split, join, and exchange operators to approach a specific position in the search space.

7.5 Approach of Other Candidate Solutions

When a particle Π_i approaches \mathcal{B}_i or \mathcal{B}_{N_i} , we ensure that the similarity of the modified partitioning \mathcal{P} and the approached partitioning $\mathcal{Q} \in \{\mathcal{B}_i, \mathcal{B}_{N_i}\}$ is increased. Recalling the definition of the similarity (see Equation (7.5)), this can, among other possibilities, be achieved by increasing $|\mathcal{P}|$, i.e., the number of partitions in \mathcal{P} , without changing $|\mathcal{P} \cap \mathcal{Q}|$ at all, or decreasing $|\mathcal{P} \cap \mathcal{Q}|$ (note that a decrease of $|\mathcal{P} \cap \mathcal{Q}|$ might come along with a decrease of $|\mathcal{P}|$). The former is obtained by splitting a partition containing elements that are members of two or more partitions in \mathcal{Q} , whereas a join or exchange achieves the latter by merging elements that reside in a single partition in \mathcal{Q} but are distributed over multiple partitions in \mathcal{P} . In contrast to random moves, the applicability of the approach operations does not only depend on \mathcal{P} 's cardinality and the size of its partitions but also on \mathcal{P} 's and \mathcal{Q} 's composition. Obviously, an approach is not possible if $\mathcal{P} = \mathcal{Q}$.

Example – Running Example of Approaching other Candidate Solutions

Unless otherwise stated, we assume that a particle at position $\mathcal{P}_* = \{\{g_1, g_2, g_4, g_5\}, \{g_3, g_6\}, \{g_7, g_8\}\}$ approaches the candidate solution $\mathcal{Q}_* = \{\{g_1, g_2\}, \{g_4, g_6, g_7\}, \{g_3, g_5, g_8\}\}$ to illustrate the operators' application in our examples (with $s_{min} = n_{min} = 2$ and $s_{max} = n_{max} = 4$). Based on the intersection $\mathcal{P}_* \cap \mathcal{Q}_* = \{\{g_1, g_2\}, \{g_3\}, \{g_4\}, \{g_5\}, \{g_6\}, \{g_7\}, \{g_8\}\}$, we have $sim(\mathcal{P}_*, \mathcal{Q}_*) = \frac{3+3}{2 \cdot 7} = \frac{3}{7}$ for the similarity between \mathcal{P}_* and \mathcal{Q}_* .

Approach Split

As stated above, the idea of the approach split operation is to extract a set of elements $L = K \cap M$ (with $|L| \geq s_{min}$) from a partition $K \in \mathcal{P}$ that contains elements of $M \in \mathcal{Q}$ as well as of one or more additional partitions in \mathcal{Q} (note that $L = K \cap M \Rightarrow L \in (\mathcal{P} \cap \mathcal{Q})$). Analogously to the definition of $\sigma_{rdm}(\mathcal{P})$, this operator can only be applied if $|\mathcal{P}| < n_{max}$. To be contained in the set of *splittable partitions* $\sigma(\mathcal{P}, \mathcal{Q})$, the partition K must also fulfill $|K| \geq 2 \cdot s_{min}$. Here, this property results from the definition of *extractable subsets* $\sigma^\uparrow(K, \mathcal{P}, \mathcal{Q})$:

$$\begin{aligned}\sigma(\mathcal{P}, \mathcal{Q}) &= \{K \mid K \in \mathcal{P} \wedge \sigma^\uparrow(K, \mathcal{P}, \mathcal{Q}) \neq \emptyset \wedge |\mathcal{P}| < n_{max}\} \\ \sigma^\uparrow(K, \mathcal{P}, \mathcal{Q}) &= \{L \mid L \in (\mathcal{P} \cap \mathcal{Q}) \wedge L \subset K \wedge |K \setminus L| \geq s_{min} \wedge |L| \geq s_{min}\}\end{aligned}$$

Because an extractable subset $L \in \sigma^\uparrow(K, \mathcal{P}, \mathcal{Q})$ is not only a *proper* subset of $K \in \mathcal{P}$ but also contained in $\mathcal{P} \cap \mathcal{Q}$, K includes, with respect to \mathcal{Q} , further elements that are not in the same partition as the elements in L . Hence, the split operator cannot be applied to approach another candidate solution if all partitions in \mathcal{P} are subsets of partitions in \mathcal{Q} , i.e., if \mathcal{P} is a refinement of \mathcal{Q} (see Equation (7.4)). For the resulting partitioning, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K\}) \cup \{K \setminus L, L\}$. Extracting the set L from K increases the similarity between \mathcal{P} and \mathcal{Q} because $|\mathcal{P}'| = |\mathcal{P}| + 1$, \mathcal{Q} is not changed, and $\mathcal{P}' \cap \mathcal{Q} = \mathcal{P} \cap \mathcal{Q}$ (i.e., the intersection of the partitionings) does not change either.

Example – Approach Split

With regard to \mathcal{P}_* and \mathcal{Q}_* , $K_* = \{g_1, g_2, g_4, g_5\}$ is the only splittable partition with extractable subset $N_* = \{g_1, g_2\}$ (N_* is, in fact, the only element of $\sigma^\uparrow(K_*, \mathcal{P}_*, \mathcal{Q}_*)$). A split results in $\mathcal{P}'_* = \{\{g_1, g_2\}, \{g_4, g_5\}, \{g_3, g_6\}, \{g_7, g_8\}\}$. As a consequence, the similarity increases from $sim(\mathcal{P}_*, \mathcal{Q}_*) = \frac{3+3}{2 \cdot 7} = \frac{3}{7}$ to $sim(\mathcal{P}'_*, \mathcal{Q}_*) = \frac{4+3}{2 \cdot 7} = \frac{1}{2}$.

Approach Join

The idea of the approach join operation is to bring together elements that are in a single partition in \mathcal{Q} but spread out across two or more partitions K, L in \mathcal{P} . As before, a join can only be applied if

$|\mathcal{P}| > n_{min}$. Similarly to the definition of *randomly* joinable partitions, *joinable partitions* $\iota(\mathcal{P}, \mathcal{Q})$ are those partitions for which *joinable counterparts* $\iota^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q})$ exist:

$$\begin{aligned}\iota(\mathcal{P}, \mathcal{Q}) &= \{K \mid K \in \mathcal{P} \wedge \iota^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q}) \neq \emptyset \wedge |\mathcal{P}| > n_{min}\} \\ \iota^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q}) &= \{L \mid L \in \mathcal{P} \wedge |K| + |L| \leq s_{max} \wedge K \neq L \\ &\quad \wedge \underbrace{\exists \mathcal{R} \subseteq \mathcal{Q} : |\mathcal{R}| \geq x}_{C_1} \wedge \underbrace{(\forall M \in \mathcal{R} : M \cap K \neq \emptyset \wedge M \cap L \neq \emptyset)}_{C_2}\} \\ &\text{with } x = 1 + \left\lfloor \frac{|\mathcal{P} \cap \mathcal{Q}|}{|\mathcal{P}| + |\mathcal{Q}|} \right\rfloor\end{aligned}$$

Please note that the definition of joinable counterparts $\iota^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q})$ is very similar to the definition of *randomly* joinable counterparts $\iota_{rdm}^{\Leftarrow}(K, \mathcal{P})$. To ensure that \mathcal{P} approaches \mathcal{Q} , we introduce additional conditions C_1 and C_2 . Condition C_2 implies $M \not\subseteq K$ because M does not only contain elements of K but also of L (with $M \in \mathcal{Q}$ and $K, L \in \mathcal{P}$). That way, we bring together elements that are in a single partition in \mathcal{Q} but in two or more partitions K, L in \mathcal{P} . Note that \mathcal{Q} cannot be approached by a join if all partitions in \mathcal{P} are supersets of partitions in \mathcal{Q} , i.e., if \mathcal{Q} is a refinement of \mathcal{P} (see Equation (7.4)). In such a situation, condition C_2 cannot be satisfied. Given that a join does not change \mathcal{Q} and yields a partitioning $\mathcal{P}' = (\mathcal{P} \setminus \{K, L\}) \cup \{K \cup L\}$ with $|\mathcal{P}'| = |\mathcal{P}| - 1$, the similarity as defined in Equation (7.5) only increases if $|\mathcal{P}' \cap \mathcal{Q}| \leq |\mathcal{P} \cap \mathcal{Q}| - x$, where $x \in \mathbb{N}_{\geq 1}$ satisfies $(|\mathcal{P}| + |\mathcal{Q}|) \cdot x > |\mathcal{P} \cap \mathcal{Q}|$ as stated in the definition of $\iota^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q})$ above. This is ensured by condition C_2 in conjunction with condition C_1 .

Example – Approach Join

With regard to \mathcal{P}_* and \mathcal{Q}_* , we have $x = 1 + \left\lfloor \frac{7}{3+3} \right\rfloor = 2$. Choosing $\mathcal{R} = \{\{g_4, g_6, g_7\}, \{g_3, g_5, g_8\}\} \subseteq \mathcal{Q}_*$ in combination with the joinable partition $L_* = \{g_3, g_6\}$ and counterpart $M_* = \{g_7, g_8\}$ satisfies conditions C_1 and C_2 (at the same time, M_* is also a joinable partition with counterpart L_*). A join results in $\mathcal{P}'_* = \{\{g_1, g_2, g_4, g_5\}, \{g_3, g_6, g_7, g_8\}\}$. Due to the resulting intersection $\mathcal{P}'_* \cap \mathcal{Q}_* = \{\{g_1, g_2\}, \{g_3, g_8\}, \{g_4\}, \{g_5\}, \{g_6, g_7\}\}$, the similarity increases from $sim(\mathcal{P}_*, \mathcal{Q}_*) = \frac{3+3}{2 \cdot 7} = \frac{3}{7}$ to $sim(\mathcal{P}'_*, \mathcal{Q}_*) = \frac{2+3}{2 \cdot 5} = \frac{1}{2}$.

Approach Exchange

If neither a split nor a join can be used to approach a partitioning $\mathcal{Q} \neq \mathcal{P}$, PSOPP falls back on the exchange operator that swaps one or more elements between a partition K contained in the set of *exchangeable partitions* $\epsilon(\mathcal{P}, \mathcal{Q})$ and one of K 's *exchangeable counterparts* $\epsilon^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q})$:

$$\begin{aligned}\epsilon(\mathcal{P}, \mathcal{Q}) &= \{K \mid K \in \mathcal{P} \wedge \epsilon^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q}) \neq \emptyset\} \\ \epsilon^{\Leftarrow}(K, \mathcal{P}, \mathcal{Q}) &= \left\{L \mid L \in \mathcal{P} \wedge K \neq L \wedge \exists M \in \mathcal{Q} : \exists \hat{K} \subset K : \right. \\ &\quad \left(\hat{K} \cap M \neq \emptyset \wedge L \cap M \neq \emptyset \wedge \hat{K} \in \mathcal{P}(\mathcal{P} \cap \mathcal{Q}) \wedge \left(\exists \hat{L} \subset L : \hat{L} \cap M = \emptyset \wedge \hat{L} \in \mathcal{P}(\mathcal{P} \cap \mathcal{Q}) \right. \right. \\ &\quad \left. \left. \wedge s_{min} \leq |(K \setminus \hat{K}) \cup \hat{L}| \leq s_{max} \wedge s_{min} \leq |(L \setminus \hat{L}) \cup \hat{K}| \leq s_{max} \right) \right\}\end{aligned} \tag{7.8}$$

Note that $\hat{K} \cap M \neq \emptyset \wedge L \cap M \neq \emptyset$ implies that $K \in \mathcal{P}$ as well as $L \in \mathcal{P}$ contain elements that belong to the same partition $M \in \mathcal{Q}$. The goal of the exchange operation is to bring these elements together. Also note that $\hat{L} \subset L$ might be an empty set \emptyset , whereas $\hat{K} \subset K$ is always non-empty. The latter causes $|\mathcal{P}|$ to be left unchanged. The condition $\hat{L} \subset L$ is implied by $L \cap M \neq \emptyset \wedge \hat{L} \cap M = \emptyset$. The reader can convince herself that excluding $\hat{L} = L$ does not restrict the applicability of the operator because the forbidden exchange of \hat{K} and $\hat{L} = L$ can be realized by swapping $K \setminus \hat{K}$ and $L \setminus \hat{L} = \emptyset$.

Integrating \hat{K} into L and \hat{L} into K increases the similarity of \mathcal{P} and \mathcal{Q} by leaving $|\mathcal{P}|$ and $|\mathcal{Q}|$ unchanged and reducing $|\mathcal{P} \cap \mathcal{Q}|$ by ≥ 1 . On the one hand, $\hat{L} \cap M = \emptyset$, $\hat{K} \cap M \neq \emptyset$, and $\hat{K} \in \mathcal{P}(\mathcal{P} \cap \mathcal{Q})$

$(\mathcal{P}(\mathcal{P} \cap \mathcal{Q}))$ denotes the power set of $\mathcal{P} \cap \mathcal{Q}$) ensure that we not only merge elements of M but also reduce the number of partitions containing elements of M by one. On the other hand, $\hat{L} \in \mathcal{P}(\mathcal{P} \cap \mathcal{Q})$ assures that we do not distribute a set of elements $V \in (\mathcal{P} \cap \mathcal{Q})$ (V is thus contained in a single partition in \mathcal{P} and \mathcal{Q}) over K and L by merging \hat{L} into K . This has to be avoided because it would decrease the similarity of \mathcal{P} and \mathcal{Q} . The conditions $s_{\min} \leq |(K \setminus \hat{K}) \cup \hat{L}| \leq s_{\max}$ and $s_{\min} \leq |(L \setminus \hat{L}) \cup \hat{K}| \leq s_{\max}$ restrict the size of the resulting partitions to the allowed range.

For the resulting partitioning, we have $\mathcal{P}' = (\mathcal{P} \setminus \{K, L\}) \cup \{(K \setminus \hat{K}) \cup \hat{L}, (L \setminus \hat{L}) \cup \hat{K}\}$. The similarity between \mathcal{P} and \mathcal{Q} is increased because $|\mathcal{P}'| = |\mathcal{P}|$, \mathcal{Q} is not changed, and $|\mathcal{P}' \cap \mathcal{Q}| \leq |\mathcal{P} \cap \mathcal{Q}| - 1$.

Example – Approach Exchange

With regard to \mathcal{P}_* and \mathcal{Q}_* , for instance, $K_* = \{g_1, g_2, g_4, g_5\}$ is an exchangeable partition with exchangeable counterparts $L_* = \{g_3, g_6\}$ and $M_* = \{g_7, g_8\}$. For example, we can exchange $\hat{K}_* = \{g_4\}$ and $\hat{L}_* = \emptyset$ between K_* and L_* by which we obtain $\mathcal{P}'_* = \{\{g_1, g_2, g_5\}, \{g_3, g_4, g_6\}, \{g_7, g_8\}\}$. Due to the resulting intersection $\mathcal{P}'_* \cap \mathcal{Q}_* = \{\{g_1, g_2\}, \{g_3\}, \{g_4, g_6\}, \{g_5\}, \{g_7\}, \{g_8\}\}$, the similarity increases from $\text{sim}(\mathcal{P}_*, \mathcal{Q}_*) = \frac{3+3}{2 \cdot 7} = \frac{3}{7}$ to $\text{sim}(\mathcal{P}'_*, \mathcal{Q}_*) = \frac{3+3}{2 \cdot 6} = \frac{1}{2}$.

However, there are situations in which the exchange operator cannot be applied: For example, consider a partitioning $\mathcal{Q}_+ = \{N_+, O_+, P_+\}$ and $s_{\min} = s_{\max} = 100$ so that each partition has a cardinality of 100. A partitioning $\mathcal{P}_+ = \{K_+, L_+, M_+\}$ cannot approach \mathcal{Q}_+ , e.g., if K_+ contains 39, 27, and 34, L_+ contains 25, 40, and 35, and M_+ contains 36, 33, and 31 elements of N_+ , O_+ , and P_+ , respectively. That is because we cannot find an \hat{L} that consists of as many elements as \hat{K} . For instance, if we choose $\hat{K}_+ \subset K_+$ with $|\hat{K}_+| = 39 + 27 = 66$, there is no combination of the above-listed subsets in L_+ or M_+ with a cardinality of 66. If we tried to integrate \hat{K}_+ into L_+ , possible cardinalities of $\hat{L}_+ \subset L_+$ would be 0, 25, 35, 40, 60, 65, 75 \neq 66. In such situations, one might relax the constraint $\hat{L} \in \mathcal{P}(\mathcal{P} \cap \mathcal{Q})$ in Equation (7.8) to $\hat{L} = U \cup V$, where $U \in \mathcal{P}(\mathcal{P} \cap \mathcal{Q})$ and $V \subset W \in (\mathcal{P} \cap \mathcal{Q})$. While this relaxation allows PSOPP to apply the operator in each situation, it only guarantees to *not decrease* the similarity of \mathcal{P} and \mathcal{Q} because we spread the elements of W over two partitions.

7.6 Evaluation and Comparison to Related Approaches

In our evaluation, we analyze PSOPP's behavior in various scenarios: We start with (1) identifying suitable values for $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}_{N_i}}$ and (2) investigating the influence of the numbers of particles on PSOPP's performance in terms of the quality of the result and the number of moves particles perform. Having identified suitable parametrizations, we (3) evaluate PSOPP's scalability with regard to different numbers of elements $|\mathcal{G}|$ to partition and (4) examine its convergence. On the basis of the partitioning constraints introduced in Section 6.2, we (5) study PSOPP's behavior in dense and sparse solution spaces. Where not stated otherwise, we make all these investigations for strict partitioning clustering (C), anticlustering (AC), and two instances of homogeneous partitioning (HPm, HPs). In a dedicated passage, we (6) compare our results to those achieved with the mathematical programming software IBM ILOG CPLEX¹ and an x-means implementation² as, to the best of our knowledge, there is no other renowned algorithm supporting more of our partitioning constraints out of the box. To examine PSOPP's performance in the context of multi-objective optimization, we (7) also consider different combinations of the objectives C, AC, HPm, and HPs.

For evaluation, we used a Java implementation of PSOPP. Each particle runs in its own thread, which allows for the parallel examination of the search space. Because preceding evaluations showed that PSOPP achieves good results with a relatively small number of particles, we used particle neighborhoods \mathcal{N}_i that contain all particles in the system, which corresponds to a full mesh topology. As mentioned in Section 7.1, \mathcal{B}_{N_i} thus represents the global best found solution \mathcal{B} . In each setting, PSOPP solved the

¹IBM ILOG CPLEX Optimizer, Version 12.4, 2011: <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, retrieved on March 1, 2016.

²Weka, Version 3.7.11, 2014: <https://sourceforge.net/projects/weka/>, retrieved on March 1, 2016.

PP for a set of elements $\mathfrak{G} = \{0, 1, 2, \dots, n-1\}$, where $n = |\mathfrak{G}|$ is the number of elements to partition. With regard to the power management case study, this can be thought of as a very heterogeneous set of power plants. Well-considered decisions are needed to create adequate partitionings, i.e., AVPPs.

As stated above, we performed evaluations for the objectives C, AC, HPm, and HPs (these types of objectives were introduced in Section 6.3). To group similar elements in case of C, the sum of the squared Euclidean distances between the elements and their cluster center (this corresponds to the “classic” k-means distance measure) is to be minimized. For AC, PSOPP *maximizes*, in accordance with [203], the sum of the squared Euclidean distances to form partitions consisting of dissimilar elements. When establishing homogeneous partitionings in case of HPm and HPs, PSOPP proceeds as described in Section 6.3: To evaluate the quality of a partitioning \mathcal{P} , PSOPP calculates a value p_K for all partitions $K \in \mathcal{P}$. As the goal is to form similar partitions, the standard deviation of the values p_K should be minimized. For HPm, p_K is the mean of the elements contained in K . In case of HPs, p_K represents the sum of the elements in K .

Hereinafter, we call the values of the aggregated squared Euclidean distances or the standard deviations *raw values* v . For better comparability, we normalize the raw values to the interval $[0, 1]$. This is achieved by the fitness function $\mathcal{F}_o(v) = 1.0 - \frac{b_o - v}{b_o - w_o}$, which yields a normalized fitness value for a specific raw value v . It is based on calculated worst w_o and best b_o values of v for objective $o \in \{C, AC, HPm, HPs\}$ (for all objectives, we calculated w_o and b_o as described in our discussion entitled “Influence of Partitioning Constraints”, which can be found in this section). For all objectives, PSOPP’s goal is to maximize the fitness since the higher the fitness, the better the solution.

With regard to the partitioning constraints, apart from $s_{min} = 2$ (i.e., each partition has to be composed of more than one element) and $n_{min} = 2$, which prevents the “grand coalition”, we did not restrict valid partitionings (that is, $s_{max} = n$, $n_{max} = \frac{n}{2}$). As discussed in Section 6.2, such restrictions enable hierarchical decomposition. The influence of other restrictions on PSOPP’s behavior is examined in a separate evaluation scenario. Where not otherwise stated, we used a time limit of 10s as termination criterion and performed 500 simulation runs for each evaluation scenario. All presented results are average values; values σ denote standard deviations.

To appraise PSOPP’s performance in all evaluation scenarios, we performed all experiments with an additional parametrization of $c_{rdm} = 1.0$, $c_{\mathcal{B}_i} = 0.0$, $c_{\mathcal{B}} = 0.0$, and 4 particles. This corresponds to a random walk through the search space. In the following, we refer to this procedure as RDM.

Identification of Suitable Parameters

First, we identified suitable parameter sets $c_{rdm}, c_{\mathcal{B}_i}, c_{\mathcal{B}}$ for C, AC, HPm, and HPs on the basis of different numbers of elements $n \in \{100, 500, 1000\}$ and particles $\#P \in \{4, 16\}$. Values for $c_{rdm}, c_{\mathcal{B}_i}$, and $c_{\mathcal{B}}$ were taken from the set $\{0.0, 0.1, 0.2, \dots, 1.0\}$. For each combination, we performed 100 simulation runs.

Figure 7.3 depicts the results of the different objectives for $\#P = 4$ and $n = 1000$. In case of C, $c_{rdm} = 0.3$, $c_{\mathcal{B}_i} = 0.0$, and $c_{\mathcal{B}} = 0.7$ turned out to be useful parameters for all combinations of n and $\#P$. The same applies to AC as well as HPs. For HPm, $c_{rdm} = 0.2$, $c_{\mathcal{B}_i} = 0.7$, and $c_{\mathcal{B}} = 0.1$ are suitable parameters. For all objectives, we observed a plateau of moderate to good fitness values for $0.0 < c_{rdm} < 0.5$, indicating the trade-off between exploration and exploitation. We further noticed that especially C profits from a higher probability of approaching \mathcal{B} than \mathcal{B}_i , whereas HPm tends to prefer the opposite. We assume that the fitness landscape of HPm contains more spikes that are worth to be explored by the particles individually, while C requires all particles to work together in order to improve a specific candidate solution (less but more prominent spikes in the fitness landscape). HPs yields high-quality results for almost every investigated parametrization. When sorting the objectives in ascending order of the importance of using an appropriate parametrization, we get HPs, AC, HPm, and C (consider the minimum and the maximum fitness values obtained for the different objectives in Figure 7.3). As we will see in the discussion of the “Influence of the Number of Elements to Partition”, this is indicative of the difficulty of solving the corresponding optimization problem. In all cases, the greater n , i.e., the problem to solve, the more important the parametrization. We used the above-mentioned parameters in our following investigations.

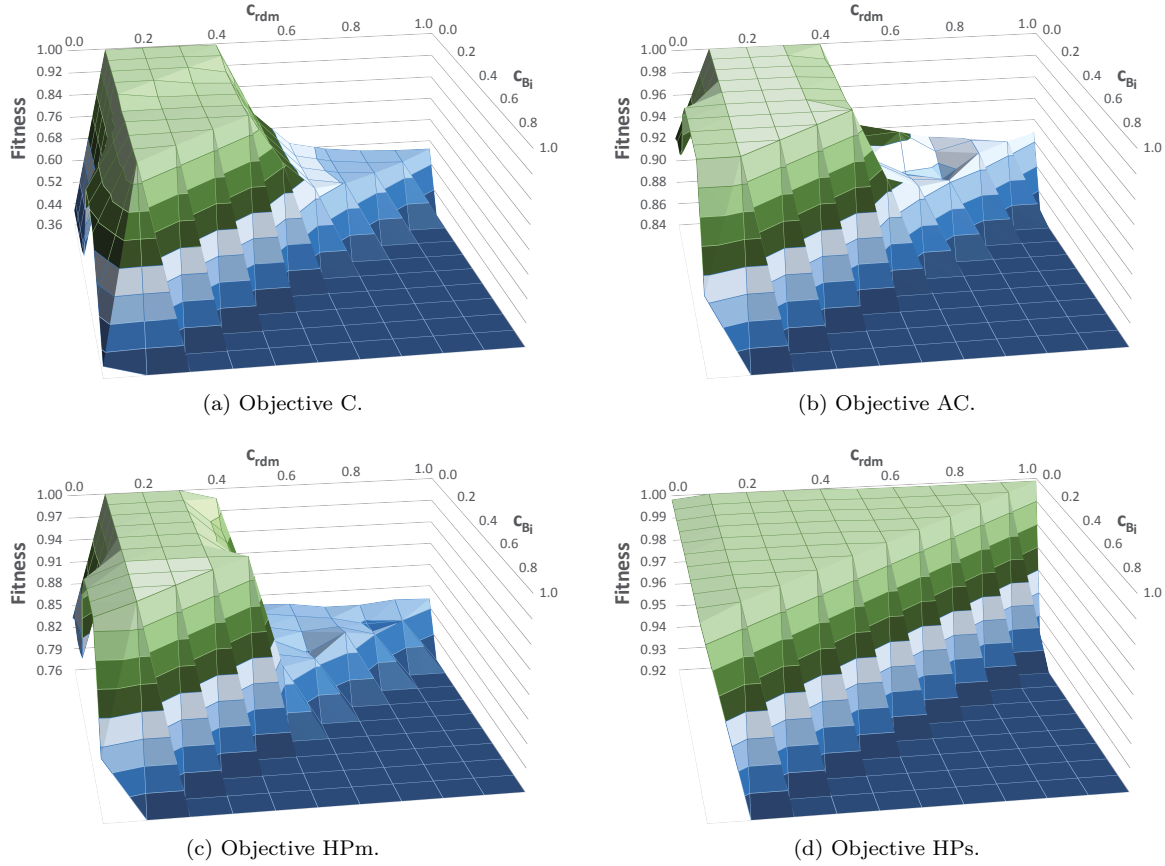


Figure 7.3: Parameter search: Fitness obtained by PSOPP for the objectives C, AC, HPm, and HPs when partitioning 1000 elements and 4 particles with different combinations of c_{rdm} , c_{Bi} , c_B . Recall that $c_B = 1 - c_{rdm} - c_{Bi}$. Results are averaged over 100 runs per parameter combination. Note that the displayed fitness ranges vary from one objective to another.

Influence of the Number of Particles

To study the effect of the number of particles $\#P$ on PSOPP's performance in terms of the quality of the result and the number of moves particles perform, we ran experiments for all combinations of $n \in \mathfrak{N}$ and $\#P \in \{2, 4, 8, 16, 32\}$ for the objectives C and HPm. Because we conducted our evaluation on 4-core Xeon machines, we expected that PSOPP does not profit much from more than 4 particles.

Indeed, we observed that the total no. of moves only shows slight increases (if not decreases) for $\#P > 4$ in case of C as well as HPm. As stated above, we attribute the threshold of 4 to our 4-core Xeon machines. In most cases, the coefficient of variation of the no. of moves per particle increases significantly with $\#P > 4$, meaning that some particles made many and others only few moves. This characteristic together with a remarkable drop of the average no. of moves per particle results in lower fitness values if the problem being solved requires a systematic exploration of the search space, as is the case with C. In case of HPm, a greater no. of particles yields better fitness values for $n > 1000$ which complies with HPm's need for exploration. Summarizing, there is certainly a trade-off between the average no. of moves per particle and the provision of diversity through a larger no. of particles that represent and improve different candidate solutions.

When comparing the results for $\#P = 2$ and $\#P \leq 4$, we see that $\#P = 4$ allows PSOPP to make 86.15% ($\sigma = 27.06\%$) or 58.18% ($\sigma = 34.67\%$) more moves in total and improves the fitness by 1.15% ($\sigma = 1.34\%$) or 2.15% ($\sigma = 2.84\%$) in case of C or HPm, respectively.

#Elements	objective C				objective AC		objective HPm		objective HPs	
	250	500	1000	2000	1000	3000	1000	4000	100	4000
w_o	1.30E6	1.04E7	8.33E7	6.67E8	250.00	750.00	499.00	1999.00	3498.76	5.66E6
b_o	62.50	125.00	250.00	500.00	8.33E7	2.25E9	0.00	0.00	0.00	0.00
RDM										
v	4.35E5 (1.50E4)	4.08E6 (9.90E4)	3.59E7 (6.51E5)	3.08E8 (4.39E6)	7.55E7 (6.70E6)	2.04E9 (1.85E8)	82.92 (40.67)	350.73 (171.56)	26.76 (1.45)	1698.47 (32.87)
$\mathcal{F}_o(v)$	0.67 (0.01)	0.61 (0.01)	0.57 (0.01)	0.54 (0.01)	0.91 (0.08)	0.91 (0.08)	0.83 (0.08)	0.82 (0.09)	0.99 (0.00)	1.00 (0.00)
PSOPP										
v	72.66 (4.15)	1423.44 (184.13)	2.51E5 (3.49E4)	7.66E7 (7.36E6)	8.33E7 (0.00)	2.25E9 (0.11)	0.00 (0.00)	75.53 (145.05)	0.66 (0.16)	1671.94 (131.18)
$\mathcal{F}_o(v)$	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.89 (0.01)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.96 (0.07)	1.00 (0.00)	1.00 (0.00)
#Partitions	121.33 (1.43)	242.65 (2.25)	481.48 (6.01)	939.75 (23.44)	2.00 (0.00)	2.00 (0.00)	2.00 (0.09)	192.31 (351.88)	38.95 (6.08)	1716.53 (227.18)
Partition Size	2.06 (0.24)	2.06 (0.24)	2.08 (0.27)	2.13 (0.34)	500.00 (312.22)	1500.00 (879.04)	499.00 (336.64)	20.80 (198.85)	2.57 (1.47)	2.33 (0.70)
#Total Moves [in 1000]	985.52 (32.24)	482.67 (18.95)	187.05 (9.33)	52.95 (5.27)	214.15 (12.98)	78.19 (3.04)	206.20 (29.37)	39.37 (17.11)	1733.64 (216.73)	36.39 (12.08)
#Rdm. Moves [in 1000]	459.54 (14.87)	222.73 (8.72)	82.97 (4.14)	20.19 (1.90)	90.29 (4.48)	31.43 (1.14)	78.84 (7.61)	15.31 (5.83)	767.87 (95.98)	11.73 (3.84)
#Appr. Moves [in 1000]	525.98 (17.43)	259.94 (10.27)	104.08 (5.23)	32.76 (3.39)	123.87 (8.52)	46.76 (2.29)	127.36 (21.96)	24.05 (11.33)	965.77 (122.44)	24.66 (8.25)
#Moves per Particle [in 1000]	246.38 (8.20)	120.67 (4.84)	46.76 (2.43)	13.24 (1.95)	53.54 (3.30)	19.55 (1.01)	51.55 (9.40)	9.84 (8.26)	433.41 (54.27)	9.10 (4.55)

Table 7.1: Selected results for the objectives C, AC, HPm, and HPs obtained with PSOPP and RDM using a time limit of 10s for different values of the number of elements. All values are averages over 500 runs. Parentheses contain standard deviations.

We provide a more detailed presentation of the results for this evaluation scenario in [15]. For the following experiments, we used $\#P = 4$ as it yields good results for C as well as HPm.

Influence of the Number of Elements to Partition

We evaluated the influence of n for the set of problem sizes $\mathfrak{N} = \{100, 250, 500, 1000, 2000, 3000, 4000\}$. As shown in Table 7.1, an increase of n comes along with a decrease in the no. of moves particles make in the search space in C, AC, HPm, as well as HPs. Evidently, that is because the application of move operators (especially the approach operators) needs more time. Because the size of the search space grows significantly with n (see Chapter 6), it is not surprising that the achieved fitness drops with greater n : While PSOPP obtains very convincing results for $n \leq 1000$ in case of C, we need a higher time limit (i.e., more than 10s) for $n \geq 2000$ (see convergence evaluation). Nevertheless, PSOPP detects that it is beneficial to establish small partitions of size two for all n . In HPm and AC, PSOPP scales much better with n . Even for $n = 4000$, the obtained fitness of 0.96 ($\sigma = 0.07$) for HPm and 1.00 ($\sigma = 0.02$) for AC is still very close to the optimum in all runs. For $n \leq 1000$, PSOPP achieves optimal results in all HPm and AC runs by establishing an appropriate partitioning consisting of two big partitions.

With regard to RDM, the obtained fitness values are remarkably lower for $n \leq 2000$ than those of PSOPP in case of C, but, although the fitness decreases, the gap narrows clearly with increasing n (from 0.43 for $n = 1000$ to 0.04 for $n = 4000$), which emphasizes the need for higher time limits. As for HPm and AC, RDM's average fitness values only show slight variations with n (on average, RDM yields 0.83 for HPm and 0.91 for AC, both with $\sigma = 0.01$) and are – while still being significantly smaller – closer to PSOPP's fitness (on average, 0.99 for HPm and 1.00 for AC, both with $\sigma = 0.01$) than in case of C. Hence, C appears to be more difficult than HPm, which, in turn, seems more difficult than AC.

HPs appears to be the easiest of our problems, which already aroused suspicion in the course of the search for appropriate parameters: Surprisingly, not only PSOPP but also RDM reaches a fitness of 1.00 for all $n \geq 250$. With regard to raw values v , PSOPP outperforms RDM by an average of 190.87 ($\sigma = 147.60$) over all $n \in \mathfrak{N}$. When minimizing instead of maximizing the fitness of HPs, the problem turns out to be much more complex: While PSOPP reaches an almost optimal average fitness value of 0.01 ($\sigma = 0.02$), RDM only yields an average of 0.94 ($\sigma = 0.06$) over all $n \in \mathfrak{N}$.

All in all, PSOPP achieves high-quality results for all objectives and a broad range of problem sizes.

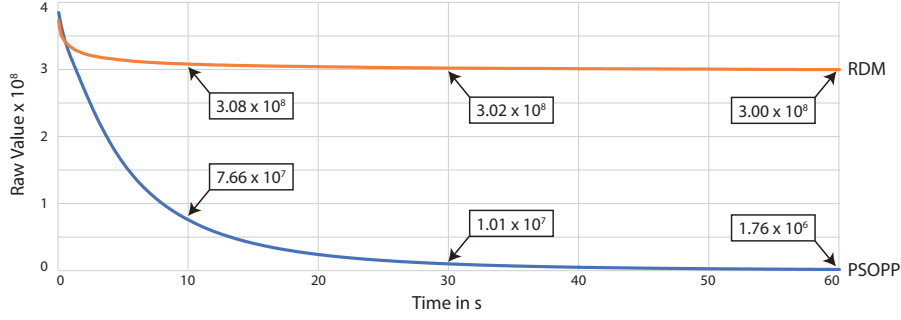


Figure 7.4: PSOPP’s and RDM’s convergence with regard to raw values for $n = 2000$, $\#P = 4$, and a time limit of 60s in case of objective C (average of 500 runs). PSOPP achieves an average fitness of 0.89, 0.98, and 1.00 after 10s, 30s, and 60s, respectively. In comparison, RDM obtains 0.54, 0.55, and 0.55.

Consequently, it is ideally suited for the creation of AVPPs whose structure adheres to the formation criteria presented in Section 6.3.

Convergence

For the evaluation of PSOPP’s convergence, we ran experiments for additional time limits of 30s and 60s for all $n \in \mathfrak{N}$. Especially objective C benefits from higher time limits in case of $n \geq 2000$: On average, the fitness is 23.49% ($\sigma = 8.70\%$) and 36.14% (16.97%) higher after 30s and 60s, respectively, compared to a limited runtime of 10s. The total no. of moves increased up to an average of 361.51% ($\sigma = 214.19\%$) after 60s. In HPm and $n \geq 2000$, PSOPP already yields high-quality results after 10s. The fitness therefore only improves by 1.57% ($\sigma = 1.43\%$) and 1.73% ($\sigma = 1.64\%$) after 30s and 60s, respectively, while the total no. of moves grows by 611.48% ($\sigma = 110.75\%$) in case of a time limit of 60s. After 60s, PSOPP achieves optimal fitness values for HPm in all runs. We observe a similar behavior in case of AC and $n \geq 2000$: The fitness can only increase by 0.12% ($\sigma = 0.17\%$) until reaching optimal values after 30s, while the total no. of moves increases up to 456.56% ($\sigma = 10.15\%$). In case of HPs, the average increase of the total no. of moves by 509.39% ($\sigma = 154.26\%$) does not have a significant effect on the fitness values since they are already at a very high level after 10s. Figure 7.4 illustrates PSOPP’s convergence in terms of the mean development of the raw value v for objective C over a time frame of 60s.

Influence of Partitioning Constraints

To examine the influence of constrained partitionings on PSOPP’s behavior, we additionally used $n_{max} = \frac{n}{2}$, $n_{min} = 0.98 \cdot n_{max}$, $s_{min} = 2$, and $s_{max} = n - (n_{min} - 1) \cdot s_{min}$ for C and HPs, and $n_{min} = 2$, $n_{max} = n \cdot 0.02$, $s_{min} = \frac{n}{n_{max}}$, and $s_{max} = \frac{n}{n_{min}}$ for AC and HPm. These parametrizations are compatible with the average number and size of partitions PSOPP found in the other evaluation scenarios (see Table 7.1): In C, it is preferred to create partitions that contain two very similar elements (e.g., a partition containing i and $i + 1$), whereas it is preferred to group two dissimilar elements i and $(n - 1) - i$ with sum $n - 1$ in order to equalize the sum of the elements of each partition in HPs. AC also favors to group such dissimilar elements to maximize the sum of the squared Euclidean distances. In our experiments, AC establishes two big partitions that contain pairs $(i, (n - 1) - i)$. Since both partitions have a mean of $\frac{n-1}{2}$, optimal results for HPm can be achieved analogously to AC (creating an individual partition for each pair $(i, (n - 1) - i)$ is an alternative to obtain optimal results for AC and HPm). That way, we also determined the objective-specific best raw values b_o needed for fitness calculations.

Note – Worst Raw Values for C, AC, HPm, HPs and $s_{min} = 2$, $s_{max} = n$

Note that the worst raw value of AC/C corresponds to the best raw value of C/AC. Regarding HPm and HPs, we calculated the worst raw values w_o as follows: For HPs, the worst case is to have two partitions of equal size, one consisting of the $\frac{n}{2}$ smallest elements and another of the $\frac{n}{2}$ largest elements. In HPm, the worst case partitioning consists of three partitions, where the first partition contains the s_{min} smallest elements, the second partition the s_{min} largest elements, and the third partition the remaining elements (the mean of the third partition is $\frac{n}{2}$).

For $n \geq 2000$, we observed that the new restrictions allow PSOPP to improve the fitness by an average of 5.66% ($\sigma = 4.04\%$) in case of C, and 1.73% ($\sigma = 1.64\%$) in case of HPm. Hence, these restrictions allowed PSOPP to achieve an optimal fitness even for $n \geq 2000$ in all HPm runs. While this improvement is accompanied by a slight average decline of the total no. of moves by 1.35% ($\sigma = 9.17\%$) in C, the total no. of moves significantly increases by 44.39% ($\sigma = 26.19\%$) in HPm. Restrictions also have a positive effect on AC: For $n = 4000$, the increase of the total no. of moves by 9.93% comes along with the ability to gain optimal fitness values in all runs. In HPs, the average raw value over all $n \in \mathfrak{N}$ can be decreased by 45.12 ($\sigma = 76.80$), although the total no. of moves drops considerably by 19.44% ($\sigma = 12.17\%$). As opposed to the other objectives, the fitness does not change. Summarizing, this shows that PSOPP cannot only deal with constrained partitionings but also benefits from them, especially if n is large. While the former is not to be taken for granted (as outlined in Section 6.5, to the best of our knowledge, there is no partitioning algorithm that supports all of these constraints out of the box), the latter is mainly because the partitioning constraints reduce the size of the search space. Where appropriate, restricting the search space is thus an alternative to raising the time limit.

Comparison with CPLEX and x-means

As stated before, we used CPLEX and an x-means implementation for comparison. We evaluated 100 runs for the following scenarios. Regarding CPLEX, we formulated the clustering and anticlustering problems as 0-1 integer programming problems. These are reminiscent of the formalization of the SPP in Equation (6.7). For $n = 100$, CPLEX obtains a solution after 30s with an average raw value of 38382.75 and 22.91 that is improved to 23014.93 and 18.10 after 60s in case of C and HPm, respectively (in all cases, $\sigma = 0.00$). PSOPP already yields far better results after 10s (29.33 with $\sigma = 2.04$ and 0.00 with $\sigma = 0.00$, respectively). Note that even RDM obtains average raw values of 19198.16 ($\sigma = 1239.23$) and 6.98 ($\sigma = 3.52$) after 10s. Please take into account that we have $w_C = 83325.00$ and $b_C = 25.00$ for the worst and the best raw values for C and $n = 100$. For HPm, we have $w_{HPm} = 49.50$ and $b_{HPm} = 0.00$. While CPLEX even needs about 720s to calculate a solution for $n = 250$, results for $n = 500$ cannot be obtained since it exceeds our 32GB of available memory. This demonstrates the problem's complexity and the need for heuristics, such as PSOPP.

Unlike PSOPP, which solves the PP in a *general manner*, x-means is specialized to problems where the objective function is additive, which implies that the costs of partitions can be assessed independently of each other. Therefore, x-means is *not* compatible with our homogeneous partitioning problems HPm and HPs. Moreover, PSOPP allows for the restriction of valid partition sizes, which is not possible in x-means. Because it is not obvious how to extend x-means by this feature, we used $s_{min} = 1$, $s_{max} = n$ to compare PSOPP to x-means (note that such a parametrization would not be feasible for creating reasonable hierarchical system structures). For these experiments, we used objective C and performed 100 x-means runs for each $n \in \mathfrak{N}$. In this situation, the highly specialized x-means obtains an average fitness of 1.00 ($\sigma = 0.00$) over all $n \in \mathfrak{N}$. With regard to the fitness $\mathcal{F}_C(v)$, PSOPP can keep up with x-means until $n = 1000$. As for raw values v , after 10s, PSOPP obtains an average of 30.57 ($\sigma = 1.29$) for $n = 100$ and 101.11 ($\sigma = 5.94$) for $n = 250$, compared to 38.00 ($\sigma = 0.00$) and 68.00 ($\sigma = 0.00$) in case of x-means (here, $b_C = 0.00$ for all $n \in \mathfrak{N}$; w_C corresponds to $8.33E4$, $1.30E6$, and $1.04E7$ for 100, 250, and 500 elements, respectively). For $n \geq 500$, we have to admit that x-means performs much better in terms of raw values than PSOPP. In case of $n = 500$, x-means reaches a raw value of 142.00

	PSOPP						x-means	
	objective HPm		objective AC		objective C		objective C	
	\bar{r}_n [in s]	\hat{r}_n [in s]	\bar{r}_n [in s]	\hat{r}_n [in s]	\bar{r}_n [in s]	\hat{r}_n [in s]	\bar{v}	
#Elements	100	0.02 (0.02)	0.16	0.07 (0.01)	0.09	0.01 (0.01)	0.08	18.00 (0.00)
	250	0.06 (0.05)	0.36	0.02 (0.02)	0.16	0.02 (0.03)	0.27	63.43 (24.22)
	500	0.21 (0.21)	0.98	0.06 (0.07)	0.31	0.08 (0.08)	0.58	604.00 (0.00)
	1000	1.11 (1.10)	4.88	0.29 (0.31)	1.51	0.41 (0.31)	1.28	5040.00 (0.00)
	2000	7.95 (8.39)	34.01	1.28 (1.68)	9.20	2.85 (2.22)	8.53	10080.00 (0.00)
	3000	28.26 (33.64)	148.02	4.18 (5.47)	27.21	9.47 (7.78)	29.80	8420.00 (0.00)
	4000	73.49 (92.60)	388.23	9.44 (14.40)	77.16	21.98 (17.31)	69.01	20160.00 (0.00)

Table 7.2: The avg. \bar{r}_n and the max. \hat{r}_n time PSOPP needed to find optimal solutions in its $500 \cdot 99\% = 495$ best runs, and the avg. raw value \bar{v} x-means obtained in its $100 \cdot 99\% = 99$ best runs limited by the time \hat{r}_n PSOPP needed to find the optimum for C ($s_{min} = n_{min} = 1$, $s_{max} = n_{max} = n$). Parentheses contain standard deviations.

on average ($\sigma = 0.00$), whereas PSOPP yields 1837.05 ($\sigma = 274.50$). A comparable value of 227.52 ($\sigma = 12.31$) is achieved after 60s.

However, we observed that PSOPP outperforms x-means when we do not constrain valid partitionings at all (here, $s_{min} = n_{min} = 1$ and $s_{max} = n_{max} = n$). Table 7.2 depicts the average \bar{r}_n and the maximal \hat{r}_n time the $500 \cdot 99\% = 495$ best runs of PSOPP needed to find optimal solutions in case of C, AC, and HPm for all $n \in \mathfrak{N}$ (please note the approximately cubic growth of \bar{r}_n and \hat{r}_n with n). In contrast to PSOPP, whose 495 best runs always yielded optimal raw values of 0.00 after \hat{r}_n seconds in C, x-means was not able to find optimal solutions. The average raw values \bar{v} obtained by x-means after \hat{r}_n seconds are also presented in Table 7.2 (analogously to PSOPP, these data are based on the $100 \cdot 99\% = 99$ best runs of x-means).

Optimization of Multiple Heterogeneous Objectives

Finally, we analyzed in which way the combination of our objectives C, AC, HPm, and HPs influences PSOPP’s ability to obtain high-quality results. For this purpose, we regarded the two three-dimensional combinations C-HPm-HPs and AC-HPm-HPs, as well as the four-dimensional case C-AC-HPm-HPs. These combinations conform to the different types of objectives needed for the creation of scalable, efficient, and stable system structures in our power management case study (see Section 6.3).

In these multi-objective optimizations, we used an a priori prioritization by taking the average of the fitness values of the corresponding optimization criteria to assess the quality of a candidate solution (each criteria was thus equally weighted). Our parameter search yielded $c_{rdm} = 0.2$, $c_{\mathcal{B}_i} = 0.1$, and $c_{\mathcal{B}} = 0.7$ for AC-HPm-HPs, and $c_{rdm} = 0.2$, $c_{\mathcal{B}_i} = 0.0$, and $c_{\mathcal{B}} = 0.8$ for C-HPm-HPs as well as C-AC-HPm-HPs. In tune with our previous observations, the valuation of the different parameter sets was mainly influenced, if not dominated, by C in C-HPm-HPs and C-AC-HPm-HPs, and by HPm in AC-HPm-HPs. Due to the multi-objective optimization, there is a conspicuous need for a systematic exploration of the search space in all cases, indicated by the high values of $c_{\mathcal{B}}$.

As for AC-HPm-HPs, the average fitness achieved for a tuple (o, n) (with $o \in \{\text{HPm}, \text{HPs}, \text{AC}\}$ and $n \in \mathfrak{N}$) did not drop by more than 0.92%, compared to optimizing for a single objective. Overall, PSOPP achieves an average fitness of 1.00 ($\sigma = 0.01$), compared to 0.90 ($\sigma = 0.00$) in case of RDM.

In scenario C-HPm-HPs, the decrease in fitness ranges between 1.55% and 12.50% for C (with an average of 5.88% and $\sigma = 4.17\%$). However, the fitness of HPm pays tribute to this relatively small reduction, in particular for large $n \geq 2000$. The drop of HPm’s fitness ranges between 1.72% and 31.85% (with an average of 13.58% and $\sigma = 13.36\%$). Again, the fitness of HPs remains at high levels and does

not diminish by more than 0.77%. Over all $n \in \mathfrak{N}$, PSOPP achieves an average fitness of 0.89 ($\sigma = 0.11$), compared to 0.73 ($\sigma = 0.04$) in case of RDM. So, the distance between PSOPP and RDM is larger than in case of AC-HPm-HPs, which we attribute to our previous observations testifying that C is more difficult than AC.

Regarding the four-dimensional case C-AC-HPm-HPs, the decline in HPm's fitness is much lower and ranges between 1.13% and 8.32% (with an average of 3.73% and $\sigma = 2.82\%$). We ascribe this to the related AC problem that is also solved in this scenario. As a result, HPm and AC weigh more than C. With respect to AC, the fitness values diminish by at least 0.34% and at most 5.25% (with an average of 1.86% and $\sigma = 1.92\%$). This comes at the price of a significant decrease in C's fitness, which ranges between 4.92% and 50.45% (with an average of 21.35% and $\sigma = 16.63\%$). Due to the supposed high density of high-quality results for HPs, its maximum decline in fitness is only 0.91%. Overall, PSOPP achieves an average fitness of 0.91 ($\sigma = 0.08$), compared to 0.72 ($\sigma = 0.02$) in case of RDM.

While one might have to adjust the weights of the different optimization criteria to the needs of a specific application, these experiments highlight PSOPP's ability to solve the PP in the context of multiple heterogeneous objectives as required in the power management case study.

Chapter Summary and Outlook

In this chapter, we introduced PSOPP, a discrete particle swarm optimizer that solves the NP-hard partitioning problem (PP) outlined in Section 6.2. In contrast to the majority of other approaches, PSOPP solves the PP in a general manner, i.e., independently of the characteristics of a specific objective function. To this end, it uses the basic set operations split, join, and exchange to explore the search space. As a result, PSOPP can be applied to diverse problems in various domains (see Section 6.5 for examples) by defining an appropriate fitness function that evaluates the quality of candidate solutions. Possible problems comprise strict partitioning clustering (with outliers), anticlustering, homogeneous partitioning, and coalition structure generation, among others. Moreover, PSOPP allows the user or the system to specify valid partitionings in terms of a minimum and maximum number and size of partitions. That way, it can be directly employed for the self-organized formation of hierarchical system structures (see Chapter 9). These properties clearly distinguish PSOPP from other partitioning methods. Our evaluation demonstrates PSOPP's versatility and shows that it finds high-quality solutions respecting prescribed partitioning constraints in different evaluation scenarios with a low number of particles.

In future work, we will extend PSOPP's approach operations so that it can directly solve multiset partitioning problems in case some of the elements to partition turn out to be identical. Since this requires a revision of the definition of the similarity of partitionings, it is not sure how these changes will influence PSOPP's performance: On the one hand, considering multisets avoids further symmetries. On the other hand, it complicates the application of approach operators. With regard to multi-objective optimization, we will allow PSOPP to gather solutions lying on the pareto frontier.

SPADA – a Decentralized Agent-Based Partitioning Algorithm

Summary. There are several approaches to deal with the complexity of solving instances of the partitioning problem (PP). These range from exploiting certain characteristics of the objective function, over using rigorous restrictions of valid partitionings (e.g., with regard to the size and number of partitions), to specific representations of the search space, among others. Another way of handling complexity are decentralized approaches, in which a number of agents solve the PP cooperatively. In this chapter, we present a decentralized partitioning algorithm, called SPADA, that solves the NP-hard PP in an iterative and regionalized manner. It is based on an overlay network the agents use to decompose the overall PP into several sub-problems in the form of regional partitionings. Because these regional partitionings not only overlap but also change from one iteration to another, their beneficial characteristics spread out across other parts, which ultimately leads to a high-quality overall solution. In contrast to many other approaches, SPADA solves the PP in a general manner. Despite its decentralized nature, it further guarantees compliance with the partitioning constraints so that it is ideally suited for the self-organized creation of scalable system structures in large-scale MAS. In our evaluation, we compare SPADA to the PSOPP algorithm in various scenarios and reveal its strengths when solving the PP in a multi-objective context. While SPADA's average solution quality ranges between 80 % and 99 % of PSOPP's in case of single objectives, SPADA outperforms PSOPP in multi-objective settings where its solutions are, on average, up to 11 % better than PSOPP's.

Publication. In this chapter, we present an advanced version of the SPADA algorithm. Its basic concepts and preliminary evaluation results have been published in Anders et al. [9].

As outlined in Chapter 6, algorithms solving variants of the PP are either (1) specialized to a particular problem in a certain domain (cf. [45, 67, 232]), (2) depend on the properties of a specific objective function (cf. [52, 159, 170]), or are (3) very restrictive with regard to the possibilities for specifying mandatory characteristics of the resulting partitioning's structure in the form of the number and the size of partitions (cf. [3, 98, 199]). On the one hand, these attributes allow the algorithms to exploit properties of the objective function, such as additivity, to represent the search space in a way that allows for a systematic search for high-quality solutions (e.g., [170]) or to reduce its size by excluding specific solutions in advance (e.g., [138]), thereby increasing the algorithms' efficiency in solving the NP-hard combinatorial optimization problem. On the other hand, they limit the algorithms' applicability, especially with regard to the self-organized formation of hierarchies. An exception to the rule is the discrete particle swarm optimizer *PSOPP* we introduced as part of this thesis in Chapter 7.

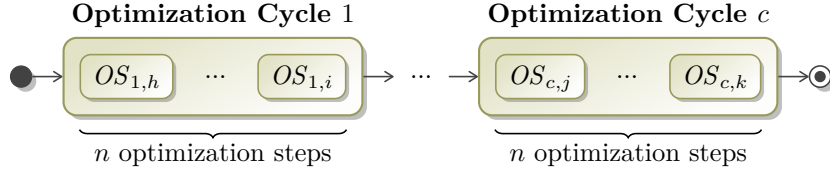


Figure 8.1: SPADA’s iterative optimization in form of optimization cycles. In every optimization cycle, each of the n agents participating in the PP performs an optimization step in which it tries to increase the fitness of its regional partitioning. The agents can perform their optimization steps either sequentially or in parallel. $OS_{c,j}$ denotes the c -th optimization step of agent a_j .

In this chapter, we present an advanced version of *SPADA*¹, which we introduced in [9].² As opposed to the afore-mentioned approaches, SPADA (1) implements a decentralized optimization scheme, (2) solves the PP in a general manner, and (3) allows for specifying suitable ranges for the number as well as the size of partitions (i.e., it ensures compliance with the partitioning constraints introduced in Section 6.2). Similar to [159] but in contrast to PSOPP and other approaches (e.g., [3, 170]), SPADA equips the agents \mathfrak{A} that have to be partitioned with the ability to solve the PP by themselves. To comply with large search spaces, SPADA applies a regionalized optimization technique in which the agents decompose the overall PP into multiple sub-problems. Each of these sub-problems corresponds to the task of optimizing the composition of a regional partitioning, that is, to improve a partial solution to the overall PP. These optimizations are performed in a regio-centralized fashion. The decomposition is obtained on the basis of an overlay network, called *acquaintances graph*. Its nodes stand for the agents participating in the self-organization process and its edges indicate the agents’ affiliation to partitions, among others. The acquaintances graph thus represents a candidate solution the agents optimize in the course of SPADA’s runtime. The idea of solving a partitioning problem on the basis of an overlay network that constitutes the candidate solution is inspired by Ogston et al. [159] (cf. the discussion of the related work in Section 6.5). As is the case with other heuristics, SPADA is an anytime algorithm. As stated in Section 6.5, this property has proved to be beneficial when solving time-consuming optimization problems that might have to be interrupted [170]. Compared to a fully localized optimization as shown in [9, 159], SPADA’s regionalized procedure reduces the attraction of local optima, enables optimization with regard to non-additive objective functions, and allows the agents to benefit from synergy effects, which is of particular interest in multi-objective optimization problems.

In SPADA, the PP is solved in an iterative manner. We refer to iterations as *optimization cycles*. In each of these cycles, every agent performs an *optimization step* in which it tries to increase the fitness of its regional partitioning (see Figure 8.1).³ Since regional partitionings are likely to change from one step to another, the overall solution results from an iterative refinement of overlapping partial solutions that is carried out from different perspectives. Its regionalized principle further allows SPADA to make selective changes with respect to the composition of an existing partitioning, which is beneficial in case of a reorganization. Due to these characteristics, SPADA can be applied to many different applications in which solving the PP is relevant. Depending on the employed objective function, SPADA can be used for homogeneous partitioning, clustering, and anticlustering, among others. Because it ensures compliance with the partitioning constraints, it is, beyond that, ideally suited for the self-organized creation of hierarchical system structures. Further details on this matter can be found in Chapter 9.

With regard to the power management case study, SPADA is used for the self-organized formation of AVPPs. Here, the main objective is to establish a homogeneous partitioning (i.e., similar AVPPs)

¹“SPADA” stands for *set partitioning in a decentralized and agent-based manner*.

²The original version of SPADA did not take the partitioning constraints into account and was not able to profit from synergy effects when creating the partitioning. These and other advantages are explained in this chapter.

³Please note that the concept of optimization cycles is not mandatory for SPADA’s functionality because it is neither necessary that the agents perform the same number of optimization steps nor that they take these steps in turns. We use it here as it helps us in conveying the idea of how the agents work together in solving the problem. For the same reason, we assume that all optimization steps within an optimization cycle are taken sequentially.

with respect to various criteria, such as the ratio between the sum of typical variations in prediction errors of non-dispatchable power plants and the sum of the rate of change of dispatchable power plants, the average costs per kWh, and the number of dispatchable power plants (see Section 6.3).

The main difference between SPADA and PSOPP is that SPADA deals with the PP's complexity by implementing a decentralized agent-based approach instead of a (centralized) population-based metaheuristic: PSOPP uses multiple particles, each representing a candidate solution, that cooperatively explore the search space by exchanging information about promising positions. Each particle has global knowledge. As opposed to that, SPADA holds only a single candidate solution. The group of agents that is to be partitioned improves this candidate solution itself by decomposing the overall PP into overlapping sub-problems whose solutions are optimized by means of regional knowledge. We can think of SPADA as a variant of a *distributed constraint optimization problem* [231], where the function that defines which agent is responsible for assigning which variables is not fixed but results from the autonomous decomposition of the PP into regional partitionings.⁴

The remainder of this chapter is structured as follows: Section 8.1 introduces the acquaintances graph that enables autonomous decomposition of the PP at runtime. SPADA's regionalized optimization method is outlined in Section 8.2. In Section 8.3, we explain how SPADA ensures compliance with the partitioning constraints. Afterwards, we show how the acquaintances graph is modified in order to improve the candidate solution by moving agents from one partition to another (see Section 8.4). Last but not least, we compare SPADA's and PSOPP's performance in various evaluation scenarios and demonstrate that SPADA outperforms PSOPP in the context of multi-objective optimization (see Section 8.5).

8.1 The Acquaintances Graph

SPADA assumes a MAS in which all agents are able to communicate with each other. In order to lower complexity when solving the PP in large-scale MAS, SPADA operates on an overlay network generated for the purpose of constraining communication and direct interactions between agents. This property of overlay networks has been studied, e.g., in peer-to-peer [137] and sensor networks [232]. In SPADA, the overlay network is a *simple directed graph*, hereinafter called *acquaintances graph*, whose nodes correspond to the agents $\mathfrak{A} = \{a_1, \dots, a_n\}$ participating in the PP. The graph's directed edges symbolize acquaintance relationships between the agents. An edge (a_i, a_j) thus indicates that a_i is acquainted with a_j .

The acquaintances graph exhibits six important properties, labeled as (P1) – (P6): Since we regard a simple graph, (P1) each edge is unique. Further, the acquaintance relation is, as there must not be any loops in simple graphs, (P2) irreflexive (i.e., an agent is not acquainted with itself) and, due to the directed edges, (P3) not symmetric. Property (P3) states that the existence of an edge (a_i, a_j) does not imply – but also not forbid – that there is an edge (a_j, a_i) . Furthermore, the acquaintance relation is (P4) not transitive so that each agent's acquaintances are limited to its outgoing edges. Although (P5) acquaintances may change over time, (P6) each agent is acquainted with the same and constant number of agents, that is, the nodes' outdegree is fixed.⁵ Note that the nodes' indegree might vary from node to node and change over time (which is why we do not consider a regular graph). As agents and partitions (in the sense of a group of agents) can only initiate interactions with their acquaintances, properties (P4) and (P6) lower and help to control the complexity of decision-making. A very simple example of an acquaintance graph is depicted in Figure 8.2a.

In SPADA, a partitioning of \mathfrak{A} is a division of the acquaintances graph into several subgraphs with a pairwise disjoint set of nodes. Each of these subgraphs stands for a partition. To specify a partitioning, edges can be marked with a partition-specific flag. A *marked edge* (a_i, a_j) between two agents a_i, a_j states that a_i is acquainted with a_j and, additionally, that a_i and a_j are members of the same partition. For the

⁴The PP can be formulated as a distributed constraint optimization problem, e.g., by using variables v_i with domains $d_i = \mathcal{P}(\mathfrak{A})$ such that each v_i represents one of the $\{n_{min}, \dots, n_{max}\}$ partitions that may be created. Here, an assignment (v_i, \emptyset) stands for a partition that is not to be formed.

⁵While the number of acquaintances per agent could also vary between a lower and an upper bound, we use a fixed number of acquaintances in our explanation for the sake of clarity.

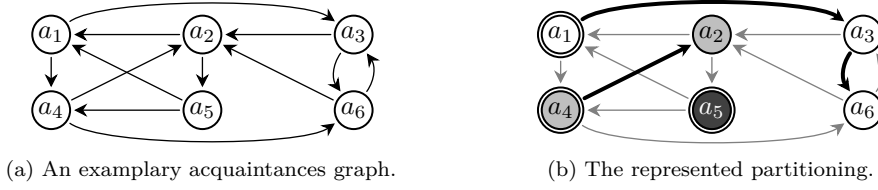


Figure 8.2: Figure 8.2a shows an acquaintances graph for a set $\mathfrak{A} = \{a_1, \dots, a_6\}$ of six agents, each of which is represented by a node. There are two acquaintances per agent, symbolized by arcs. Agent a_6 , for instance, is acquainted with a_2 and a_3 . Figure 8.2b depicts a partitioning of \mathfrak{A} consisting of the three color-coded partitions $\{a_1, a_3, a_6\}$, $\{a_2, a_4\}$, and $\{a_5\}$. Their leaders a_1 , a_4 , and a_5 are indicated by double circles. Light thin and dark bold arcs denote unmarked and marked edges, respectively.

sake of clarity, we call edges without such a mark *unmarked edges* in the following. The acquaintances graph thus represents a *candidate solution* that is optimized in the course of SPADA's runtime.

Strictly speaking, each partition is an *arborescence* (i.e., a directed rooted tree whose edges point away from the root) of marked edges. As such, the overall partitioning corresponds to a directed forest. This means that all operations SPADA performs to come to a solution can be mapped to graph operations applied to the overlay network. As outlined in Section 8.4, the arborescence property of partitions simplifies maintaining the properties of the acquaintances graph when modifying partitions.

Each partition always has a designated agent, called *leader*, that is known to each partition member. A leader unambiguously identifies its partition. We define that it is given by the corresponding arborescence's root. With a an arbitrary agent and \mathcal{R}_{me}^* the reflexive transitive closure of the binary relation \mathcal{R}_{me} induced by marked edges, the set $K = \{a \mid l(K) \mathcal{R}_{me}^* a\}$ constitutes the partition K of leader $l(K)$. Hence, two agents can be members of the same partition without a marked edge between them. Figure 8.2b shows an example of a partitioning grounded on an acquaintances graph.

The main task of each leader is to periodically change the acquaintances of its partition members and, most importantly, to optimize the composition of a part of the overall partitioning according to application-specific formation criteria. This part is restricted to the leader's own as well as its *acquainted partitions*. Acquainted partitions are those partitions containing acquaintances of the leader's partition members. Therefore, only the members' unmarked edges are relevant for obtaining the acquainted partitions. Based on the function $partition(a_j)$ that identifies the partition of an agent a_j and the function $acqAgents(a_i)$ that gathers the acquaintances of a partition member a_i , the set of acquainted partitions of a partition K is thus defined as $acqPartitions(K) = \{partition(a_j) \mid a_i \in K \wedge a_j \in acqAgents(a_i)\} \setminus K$ (for partitions, the acquaintance relation is irreflexive, too). With regard to Figure 8.2b, the white partition $\{a_1, a_3, a_6\}$ is only acquainted with the gray partition $\{a_2, a_4\}$ due to its unmarked edges (a_1, a_4) , (a_3, a_2) , (a_6, a_2) ; the unmarked edge (a_6, a_3) has no influence as a_3 is also a member of the white partition. Limiting a leader's changes to a part of the entire partitioning restricts the size of its (regional) search space and thereby reduces its computational cost of finding suitable modifications. That way, the acquaintances graph is used to decompose the overall PP into multiple sub-problems. Although every agent is capable of being a leader, there is only one leader per partition to avoid inconsistencies in the course of the formation process. Consequently, non-leaders show a rather passive behavior until they become leaders themselves.

Because partitions located in separated subgraphs cannot exchange agents, SPADA is initialized with a weakly connected acquaintances graph (i.e., the graph's undirected counterpart is connected). In Section 8.4, we show that the graph operations modifying the partitions and acquaintances do not break the weak connectivity. This prevents SPADA from not being able to reach specific positions in the search space. Figure 8.3 depicts a more extensive example of an acquaintances graph. We will use this structure to illustrate the SPADA algorithm in the following sections.

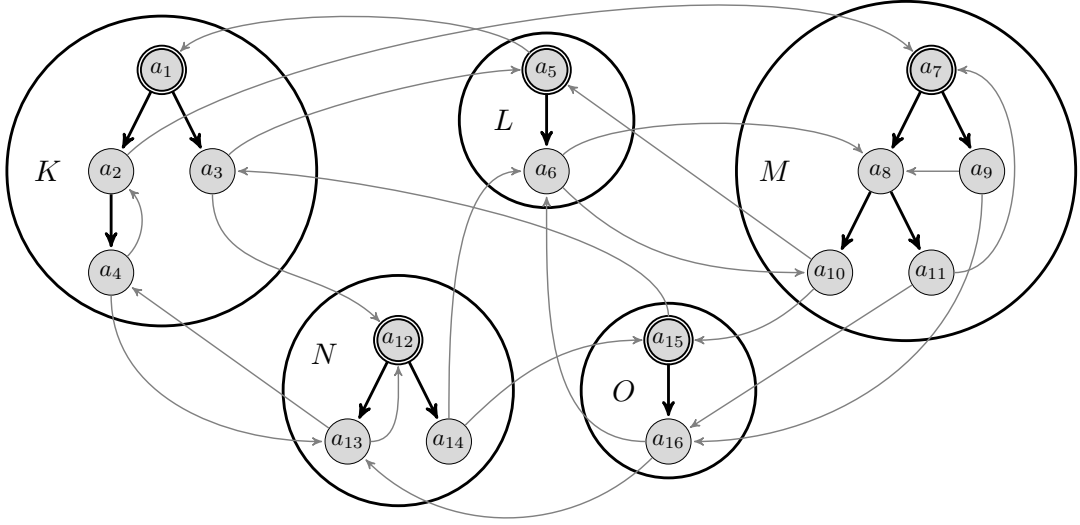


Figure 8.3: An acquaintances graph for a set $\mathfrak{A} = \{a_1, \dots, a_{16}\}$ of sixteen agents, each having two acquaintances. The partitioning is composed of five partitions $\{K, \dots, O\}$, represented by the five larger circles. The partitions' leaders are indicated by double circles. Light thin and dark bold arcs denote unmarked and marked edges, respectively.

8.2 Decentralized Formation of Partitions

SPADA features two important and interconnected characteristics that foster its ability to find high-quality solutions for multi-objective optimization problems in large search spaces. These characteristics go hand in hand with its decentralized nature:

Regionalized Optimization. Leaders do not only try to optimize the composition of their own partition but of a *region*, i.e., a regional partitioning, comprising their own as well as acquainted partitions. Consequently, leaders perform a *regionalized* in lieu of a *fully local* optimization.

Concept of Transactions. One possibility for optimizing the partitioning would be to allow each leader to make incremental changes to its regional partitioning. In such an approach, a leader would move a partition member to an acquainted partition or vice versa independently of other possible changes. Instead, leaders determine suitable combinations of moves, so-called *transactions*, that are carried out atomically. By restricting each leader's transactions to moves that involve its own partition, SPADA promotes agent autonomy rather than heteronomy. Since this measure constrains feasible moves, it also reduces the size of the leader's regional search space, which simplifies the problem of creating suitable transactions.

There are several advantages that accrue from these properties: In a fully local optimization approach in which leaders create transactions on the basis of their local knowledge, the leader receiving a proposal for a transaction is likely to have a different opinion concerning the transaction's valuation than the proposer. If the disagreeing leader expects a decline in the partitioning's fitness, it might abort the transaction. SPADA's regionalized principle prevents such disagreements, which is beneficial as a higher-than-necessary number of aborted transactions causes communication overhead. Furthermore, the concept of transactions is a flexible instrument allowing SPADA to examine the search space in a more efficient and effective way. In comparison to individual changes, transactions do not only decrease the algorithm's runtime but also allow for avoiding local optima. For instance, although both moving agent a_1 from partition K to partition L or agent a_2 from partition M to K might decrease the fitness of the regarded region, moving a_1 and a_2 together might actually increase its fitness. This example also shows

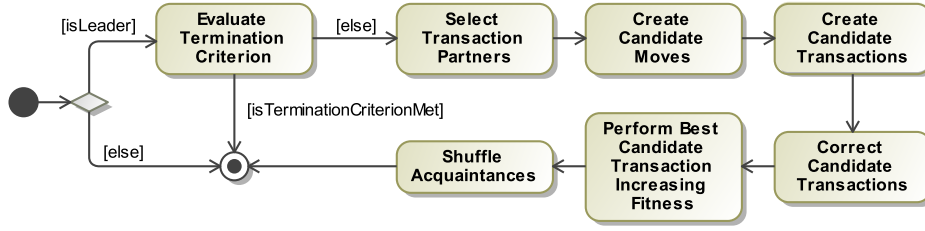


Figure 8.4: Actions leaders perform during their optimization step in each optimization cycle.

that the concept of transactions goes hand in hand with the regionalized principle: Although performing the transaction of moving a_1 and a_2 might indeed worsen the composition of partitions L and M , it might, at the same time, improve the composition of partition K to such an extent that the region's overall fitness increases. In a nutshell, the concept of transactions and the regionalized optimization allow SPADA to benefit from synergy effects. These are especially advantageous in multi-objective optimization problems, as is the case with power systems where partitionings should be formed with respect to numerous criteria (see Section 6.3). As opposed to this, a fully local optimization or an approach relying on individual changes is likely to struggle to find single modifications that improve the fitness among different criteria. In the worst case, such an approach would not be able to proceed with the optimization. In the following subsections, we sketch SPADA's decentralized approach to solve the PP.

Basic Procedure of SPADA

Once a reorganization is triggered, SPADA starts with the initialization of the acquaintances graph for the agents \mathfrak{A} participating in the PP. This encompasses (1) the creation of arborescences representing the partitions, i.e., the (sub)systems, of the partitioning that has to be reorganized as well as (2) the randomized yet guided generation of unmarked edges. The latter ensures the acquaintances graph's weak connectivity as demanded in Section 8.1. In case SPADA is used to create an initial partitioning, it is initialized with a randomly generated partitioning. We assume that SPADA is always initialized with a partitioning that satisfies the partitioning constraints. Section 6.4 explains how this is achieved in case of a reorganization as well as for the creation of an initial partitioning.

Subsequent to the initialization, each agent performs an optimization step in each optimization cycle. As already mentioned in Section 8.1, non-leaders exhibit a passive behavior and thus finish their optimization step without performing any action. Figure 8.4 shows the main activities each leader's optimization step comprises in order to improve the partitioning's fitness: If the chosen termination criterion is not met, the regarded leader first determines a regional partitioning to optimize by choosing a set of transaction partners from its set of acquainted partitions. Afterwards, it creates candidate moves, i.e., possible modifications, for its regional partitioning to optimize. Several candidate moves are then combined to candidate transactions. The concept of candidate transactions allows SPADA to profit from synergy effects originating from changing the affiliation of two or more agents at once. While candidate transactions are created in a target-oriented manner such that their application would increase the region's fitness, the procedure does not guarantee that the resulting partitioning would satisfy the partitioning constraints. The sacrifice of creating feasible candidate transactions reduces complexity and fosters SPADA's exploratory behavior. However, since the partitioning ultimately has to comply with the partitioning constraints, the leader corrects the candidate transactions after their creation. To be able to solve the PP – despite the global minimum- and maximum-number-of-partitions constraints – in a decentralized manner, SPADA decomposes the global constraints into local counterparts whose satisfaction implies adherence to the global constraints. Apart from a one-time centralized correction of the partitioning needed to decompose (i.e., localize) the global constraints, all corrections can thus be made locally. Having repaired all candidate transactions, the leader applies the best candidate

transaction that increases the region's fitness to the acquaintances graph. This means that the changes stipulated in the candidate transaction are transferred to the underlying graph structure representing the current partitioning. The global number-of-partitions constraints are localized by providing each leader with upper bounds for creating and resolving partitions that are updated in accordance with the applied candidate transaction. Finally, the leader shuffles the acquaintances of its partition members so that regional partitionings quite certainly change from one optimization step to another. This allows regional solutions to evolve over time and their beneficial characteristics to spread out across wider parts of the overall partitioning. The overall solution is thus the result of an iterative refinement of overlapping partial solutions that is carried out from different perspectives. Note that a concurrent implementation of this procedure in which multiple overlapping regional partitionings might optimize themselves at the same time has to rule out conflicting transactions, e.g., by “locking” partitions taking part in a regional optimization.

In the following, we describe these steps in greater detail. SPADA's approach to satisfy the partitioning constraints, including the localization of the global minimum- and maximum-number-of-partitions constraints as well as the correction of candidate transactions, is outlined in Section 8.3. Section 8.4 explains how the acquaintances graph is modified in order to comply with the changes stipulated in the selected candidate transaction on the one hand and the graph's properties introduced in Section 8.1 on the other hand.

Termination, Soundness, and Completeness

Similar to PSOPP (see Section 7.2), SPADA provides several termination criteria ranging from a predefined amount of time or number of optimization cycles to a predefined minimal fitness value. Also SPADA expects that the termination criterion is formulated as a conjunction of hard constraints. Other application-specific termination criteria that are in line with the corridor of correct behavior can therefore be easily integrated into the algorithm. As stated in Section 7.2, we limit the runtime of a reorganization in the power management case study to ensure that the AVPPs have enough time to recalculate the schedules.

If a centralized perspective is necessary to decide whether to terminate or not, a dedicated leader checks if the termination criterion is satisfied at the beginning of each optimization cycle. In case the number of partitions is restricted, the same leader also takes care of the one-time centralized correction that ensures compliance with the global number-of-partitions constraints (see Section 8.3). If this correction has not yet been performed, the dedicated leader repairs the overall partitioning before returning it as solution. Due to that and because the satisfaction of the partition-size constraints is ensured during the entire optimization process, SPADA always returns a feasible solution, which is why SPADA is sound. However, since SPADA is a heuristic, it is not complete in a general sense. Basically, this can be ascribed to its decentralized regional optimization scheme. While SPADA is complete if feasible solutions are only restricted by the partitioning constraints, it cannot guarantee to find solutions exhibiting other mandatory properties, such as a predefined minimal quality. Note that the same applies to PSOPP.

Problem Decomposition through Regionalized Optimization

To decompose the overall PP into multiple sub-problems, SPADA makes use of the acquaintances graph. Each sub-problem corresponds to the task of optimizing the composition of a *regional partitioning*, i.e., a part of the overall partitioning. Each regional partitioning is optimized by a specific leader. The set of partitions contained in a leader's regional partitioning depends on the acquaintances of the members of its partition K_{opt} . With respect to a single optimization cycle, the overall partitioning is thus regarded and optimized from different perspectives.

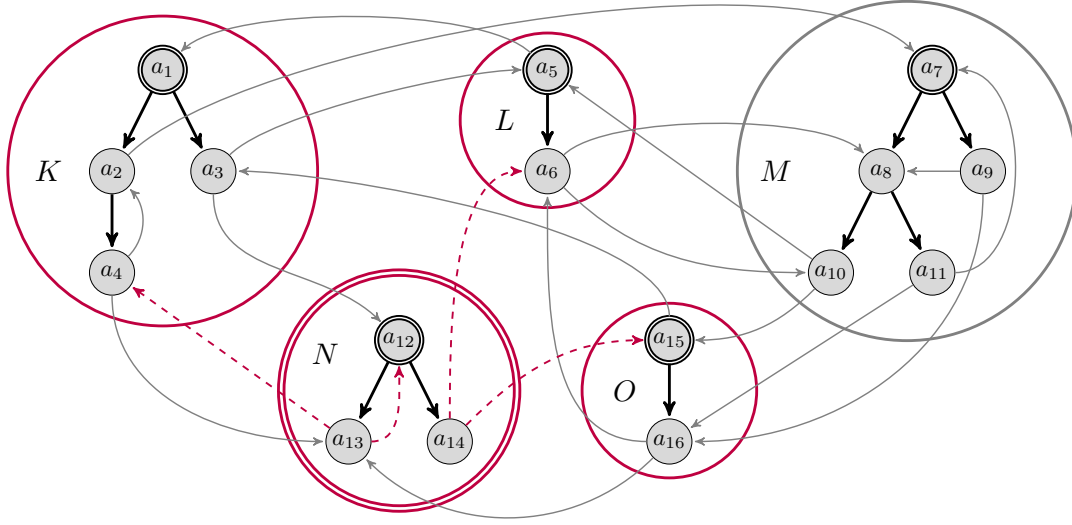


Figure 8.5: An alternative view onto the example given in Figure 8.3: The regional partitioning $\mathcal{R} = \{K, L, N, O\}$ of partition N is highlighted by the large dark circles. Marked edges are symbolized by the bold solid arcs. All other arcs denote unmarked edges. The unmarked edges of N are accentuated by the dashed arcs. A partition's regional partitioning is defined by the heads of its unmarked edges and always contains the partition itself. Therefore, N would also be in \mathcal{R} if the unmarked edge (a_{13}, a_{12}) did not exist.

Definition 6 (Regional Partitioning). The **regional partitioning** \mathcal{R} of a leader consists of its own partition K_{opt} as well as its acquainted partitions $\text{acqPartitions}(K_{\text{opt}})$:

$$\mathcal{R} = \text{acqPartitions}(K_{\text{opt}}) \cup \{K_{\text{opt}}\}$$

Because of the constant number of acquaintances per agent $\#ACQ$, the number of acquainted partitions is limited by the number of partition members $|K_{\text{opt}}|$: $\text{acqPartitions}(K_{\text{opt}}) \leq \#ACQ \cdot |K_{\text{opt}}| - (|K_{\text{opt}}| - 1)$. With regard to partition N in Figure 8.5, we have $\text{acqPartitions}(N) = \{K, L, O\}$ for N 's acquainted partitions, resulting from the unmarked edges (a_{13}, a_4) , (a_{14}, a_6) , (a_{14}, a_{15}) , and thus $\mathcal{R} = \{K, L, N, O\}$ for its regional partitioning.

Each leader is responsible for optimizing the composition of its regional partitioning according to the application-specific formation criteria specified in the fitness function. This is achieved by moving agents from its own partition to acquainted partitions or vice versa, or creating a new partition by excluding some of its partition members. The latter can be beneficial if the partition's or the corresponding agents' properties have changed in a way that the formation criteria no longer favor their inclusion. The regionalized principle allows SPADA to make selective changes with respect to the composition of an existing partitioning. This characteristic is particularly beneficial when SPADA is used to reorganize the structure of a (sub)system. In order to pursue a common objective, the decentralized optimization scheme requires all agents to use the same fitness function. Of course, it also has to comply with the function we finally use to evaluate SPADA's result. For instance, if the goal is to obtain a homogeneous partitioning, the leaders should also perform a homogeneous partitioning in their region. Due to the decentralized approach, leaders use regional instead of global knowledge, that is, knowledge of their regional partitioning, for optimization in order to limit communication efforts. Furthermore, global information would be outdated as soon as one partition changes its regional partitioning, which decreases

the information's significance in case of a concurrent optimization of disjoint regional partitionings.

As discussed in Chapter 6 and demonstrated in Section 7.6, the characteristics of the employed fitness function dictate the complexity of the PP and thus the algorithm's performance. This situation is aggravated when solving the PP in a decentralized manner because it depends on the fitness function whether the decomposition of the overall PP into sub-problems yields regional partitionings that can be optimized *independently* of other disjoint regional partitionings. Basically, such a decomposition is possible if the fitness function uses only local information to assess the quality of a partition, or, in other words, if it is additive (i.e., the partitioning's quality is the sum of the quality of each included partition, and a partition's quality does not depend on the composition of the other partitions). In case of the "classic" k-means distance measure that is used in clustering and anticlustering, for example, a regional partitioning can be optimized and assessed independently of the others. That is because, in the k-means distance measure, the costs of the individual partitions stem from the dissimilarity of their composition, which is evaluated locally without taking account of the other partitions (see Equation (6.4)). This implies that the partitions' costs are additive, which is why the overall costs change by the same value the regional optimization changes the costs of the regional partitioning. Since simultaneous optimizations of different disjoint parts of the overall partitioning do not interfere with each other, leaders can draw exact conclusions from possible composition adjustments. Regarding homogeneous partitioning, the objective is to establish similar partitions (see Section 6.3). In the corresponding fitness function, this is expressed by calculating the standard deviation of all partitions with respect to a specific property, such as the mean predictability of partition members. The costs originating from a single partition therefore depend on the composition of the other partitions, which is why the non-interference property does not hold for homogeneous partitioning. As opposed to the k-means distance measure, the fitness function is thus not additive. Since the evaluation of a partition's costs would require global information, there is no guarantee that the regional optimization reduces the costs of the overall partitioning. For this reason, a leader can only assume that its regional partitioning exhibits the basic characteristics of the overall partitioning and is thus an appropriate representative. The quality of this approximation depends on the regional partitioning's size. If it is too small, it can only give a rough indication of the quality of its partitions. In such a situation, increasing the similarity of the partitions within the regional partitioning can actually decrease the similarity of the overall partitioning. Consequently, one has to make a compromise between the significance of the regional partitioning and the size of the regional search space.

Example – Decentralized Optimization and Homogeneous Partitioning

Let us regard a partitioning \mathcal{P} , currently consisting of five partitions $\{K, L, M, N, O\}$, each containing four power plants. In this setting, the goal is to create a partitioning in which each partition has the same average rate of change (according to Section 6.3, this corresponds to a homogeneous partitioning mean). In the current partitioning, all power plants in K, L, M, N , and O have a rate of change of 0, 2, 3, 4, and 6, respectively. So the current average rate of change of the partitions is also 0, 2, 3, 4, and 6. Hence, the standard deviation of the overall partitioning is currently 2. Let us assume that L performs an optimization of its regional partitioning $\mathcal{R} = \{L, M, N\}$ whose standard deviation is approximately 0.8. The following two regional optimizations illustrate that the fitness of the overall partitioning does not necessarily has to increase with the fitness of the regional partitioning:

1. By exchanging two power plants between L and N , L 's leader can optimize the composition of its regional partitioning: Since the resulting regional partitioning consist of three partitions, each with an average rate of change of 3, the leader decreased the standard deviation of the regional partitioning to 0. These changes also increase the fitness of the resulting overall partitioning as the standard deviation is now approximately 1.9.
2. By merging partitions L and N in \mathcal{R} , the average of the resulting partition L' is 3 and the standard deviation of the resulting regional partitioning again 0. However, the overall fitness deteriorates because the standard deviation of the resulting overall partitioning \mathcal{P}' rises to approximately 2.1.

SPADA’s potential of performing an iterative optimization from different perspectives becomes evident in the second case. Given the new overall partitioning $\mathcal{P}' = \{K, L', M, O\}$, the leader of partition K with regional partitioning $\{K, M, O\}$ could now exchange two power plants between partitions K and O in order to make the partitions in its regional partitioning more similar. In this case, the new resulting overall partitioning consist of four partitions, each with an average rate of change of 3. Despite the decentralized optimization, the procedure achieves an optimal standard deviation of 0.

There are several characteristics that promote SPADA’s ability to find high-quality solutions despite its decentralized nature: With regard to a specific optimization cycle, it is very likely that the partitions’ regional partitionings partly overlap. In Figure 8.5, for instance, both partition N as well as partition M are acquainted with partition O , and, in fact, all partitions are acquainted with partition L . Assessing and revising the composition of partitions from different perspectives alleviates the effect of possibly imprecise approximations, reduces the risk of getting stuck in local optima, and fosters certain characteristics of promising partial solutions to have a positive influence on the overall solution by diffusing into other parts of the overall partitioning. These effects are intensified by (1) agents moving from one partition to another (since these bring along their acquaintances, regional partitionings are likely to change from one optimization step to another), (2) the decision that leaders shuffle the acquaintances of their members at the end of each optimization step, and (3) the iterative optimization over multiple optimization cycles. The two former points also promote exploration as varying acquaintances in the form of agents and partitions enriches the variety of potential changes.

In this process, the acquaintances graph restricts the size of the regional search space and thus controls the degree of decomposition. With a fixed number of acquaintances per agent, the degree of decomposition tends to increase with the number of agents participating in the PP. To further reduce the size of the regional search space and thereby the leader’s computational cost of finding suitable candidate transactions (i.e., combinations of partition modifications), the leader determines a non-empty set of *transaction partners* (TP) it plans to exchange agents with. The set of selected transaction partners is therefore not only a subset of its acquainted partitions but also of its regional partitioning. The set specifies the so-called *regional partitioning to optimize*.

Definition 7 (Regional Partitioning to Optimize). *A leader’s **regional partitioning to optimize** \mathcal{R}_{opt} is a subset of its regional partitioning \mathcal{R} , consisting of its own partition K_{opt} and $\#TP - 1$ other partitions:*

$$\mathcal{R}_{\text{opt}} \subseteq \mathcal{R} \wedge K_{\text{opt}} \in \mathcal{R}_{\text{opt}} \wedge |\mathcal{R}_{\text{opt}}| = \#TP$$

The regional partitioning to optimize is randomly created by selecting $\#TP - 1$ acquainted partitions using a uniform distribution. The parameter $\#TP$ is used to control the size of the regional partitioning to optimize dependent on the size of the leader’s partition. Larger partitions are therefore provided with more options in terms of transaction partners than smaller partitions, which allows them to optimize larger regions. The randomized selection of transaction partners spurs exploration.

With respect to Figure 8.5, let us assume that $\#TP = 3$ and that the leader of partition N selects the partitions K and L as transaction partners. As such, we have $\mathcal{R}_{\text{opt}} = \{K, L, N\}$ for N ’s regional partitioning to optimize.

Creation of Candidate Transactions

Having determined the regional partitioning to optimize, a leader creates a set of *candidate moves*. Each candidate move represents a possible modification of the leader’s regional partitioning. A candidate move proposes either (1) to transfer a specific agent from the leader’s partition to another partition in \mathcal{R}_{opt} or vice versa, or (2) to exclude one of the leader’s partition members. In the latter case, the excluded agent becomes a member of a new partition that currently does not exist.

Definition 8 (Candidate Move). A **candidate move** cm is a triple $(a, K_{\text{from}}, K_{\text{to}})$ stating that agent a should be moved from its current partition K_{from} to another partition K_{to} . With respect to the regional partitioning to optimize \mathcal{R}_{opt} , we distinguish the two disjoint sets of **candidate moves for integration** CM_{in} and **candidate moves for exclusion** CM_{out} :

$$\begin{aligned} CM_{\text{in}} &= \{(a, K_{\text{from}}, K_{\text{to}}) \mid a \in K_{\text{from}} \wedge K_{\text{from}} \in \mathcal{R}_{\text{opt}} \setminus \{K_{\text{opt}}\} \wedge K_{\text{to}} = K_{\text{opt}}\} \\ CM_{\text{out}} &= \{(a, K_{\text{from}}, K_{\text{to}}) \mid a \in K_{\text{from}} \wedge K_{\text{from}} = K_{\text{opt}} \wedge K_{\text{to}} \in \mathcal{R}_{\text{opt}} \setminus \{K_{\text{opt}}\} \cup \{\emptyset\}\} \end{aligned}$$

The set CM_{in} contains all moves transferring a member of an acquainted partition into K_{opt} . By contrast, CM_{out} includes all moves transferring a member of K_{opt} into either an acquainted partition or a new partition, indicated by $K_{\text{to}} = \emptyset$, that currently does not exist.

There are always $\left| \bigcup_{K_{\text{from}} \in \mathcal{R}_{\text{opt}} \setminus \{K_{\text{opt}}\}} K_{\text{from}} \right|$ candidate moves in CM_{in} and $|K_{\text{opt}}| \cdot |\mathcal{R}_{\text{opt}}|$ candidate moves in CM_{out} . By excluding candidate moves that do not affect the leader's partition K_{opt} , SPADA mitigates heteronomy of the acquainted partitions $\text{acqPartitions}(K_{\text{opt}})$ and thus promotes agent autonomy. As we will discuss after the following definition of candidate transactions, this restriction also considerably reduces the size of the regional search space, which simplifies the leader's search for suitable changes.

With regard to Figure 8.5 and our exemplary regional partitioning to optimize $\mathcal{R}_{\text{opt}} = \{K, L, N\}$, the leader of partition N obtains $CM_{\text{in}} = \{(a, K, N) \mid a \in \{a_1, a_2, a_3, a_4\}\} \cup \{(a_5, L, N), (a_6, L, N)\}$ and $CM_{\text{out}} = \{(a, N, P) \mid a \in \{a_{12}, a_{13}, a_{14}\} \wedge P \in \{K, L, \emptyset\}\}$ as sets of candidate moves for integration and exclusion. The tuple (a_1, K, L) is not a valid candidate move as it involves neither the leader's partition N nor the empty set \emptyset .

Leaders combine multiple candidate moves to *candidate transactions*, each specifying in which way the regional partitioning to optimize \mathcal{R}_{opt} could be changed in order to increase its fitness.

Definition 9 (Candidate Transaction). A **candidate transaction** ct is defined as a set of **conflict-free** candidate moves $cm \in CM_{\text{in}} \cup CM_{\text{out}}$. Two candidate moves $(a, K_{\text{from}}, K_{\text{to}}), (a', K'_{\text{from}}, K'_{\text{to}}) \in CM_{\text{in}} \cup CM_{\text{out}}$ are considered conflict-free if they regard different agents, i.e., if $a \neq a'$.

With regard to our running example and Figure 8.5, $ct = \{(a_4, K, N), (a_{13}, N, \emptyset), (a_{14}, N, \emptyset)\}$ is a conflict-free candidate transaction, whereas $ct' = \{(a_{14}, N, K), (a_{14}, N, L)\}$ contains conflicting candidate moves. Because partitions are disjoint, CM_{in} never contains conflicting candidate moves. By contrast, there are $|\mathcal{R}_{\text{opt}}|$ conflicting candidate moves for each member of K_{opt} in CM_{out} .

The changes a leader can implement during an optimization step are limited to the application of a single candidate transaction. This candidate transaction is chosen from the regional search space, which is defined as the set of all possible resulting regional partitionings. If we did not restrict possible modifications of \mathcal{R}_{opt} to moves involving K_{opt} , it would consist of all possible partitionings that can be formed with the agents in \mathcal{R}_{opt} . With $x = |CM_{\text{in}}| + |K_{\text{opt}}|$ the number of agents in \mathcal{R}_{opt} , the size of the regional search space would thus be given by the x th Bell number \mathcal{B}_x (cf. Chapter 6). Limiting the modifications to conflict-free combinations of candidate moves from the sets CM_{in} and CM_{out} significantly decreases the size of the regional search space to $|\mathcal{P}(CM_{\text{in}})| \cdot (|\mathcal{R}_{\text{opt}}| + 1)^{|K_{\text{opt}}|}$ (all subsets of the candidate moves for integration can be combined with the member-specific decisions whether a member of K_{opt} should stay, be transferred into an existing partition, or be moved into the new partition). By itself, the decision to forbid leaders to create more than one new partition per optimization step decreases the number of possibilities to partition K_{opt} into new partitions from $\mathcal{B}_{|K_{\text{opt}}|} - 1$ to $|\mathcal{P}(K_{\text{opt}}) \setminus \{\emptyset, K_{\text{opt}}\}|$ (for $|K_{\text{opt}}| = 10$, for instance, there are thus only 1022 instead of 115974 possible partitionings of K_{opt} into new partitions). This is of particular interest if partitions have to consist of more than one member (i.e., if $s_{\text{min}} > 1$): The greater the minimum size of partitions, the more unlikely the creation of a new partition that satisfies the minimum-size constraint. But the less new partitions can be created, the more likely it is that they are large enough.

Algorithm 1 Calculation of the Resulting Partitioning**Require:** \mathcal{P} is a partitioning, ct is a (conflict-free) candidate transaction.**Ensure:** \mathcal{P}' is the partitioning that would result from applying all candidate moves in ct .

```

1: procedure DETERMINERESULTINGPARTITIONING( $\mathcal{P}$ ,  $ct$ )
2:    $\mathcal{P}' \leftarrow \emptyset$  ▷ holds the resulting partitioning
3:   ▷ iteratively create the resulting partitioning  $\mathcal{P}'$ 
4:   for all  $K \in \mathcal{P} \cup \{\emptyset\}$  do ▷  $\emptyset$  represents the possibly newly created partition
5:      $L \leftarrow \{a \mid (a, K_{\text{from}}, K_{\text{to}}) \in ct \wedge K_{\text{from}} = K\}$  ▷ all agents that are removed from  $K$ 
6:      $M \leftarrow \{a \mid (a, K_{\text{from}}, K_{\text{to}}) \in ct \wedge K_{\text{to}} = K\}$  ▷ all agents that are transferred into  $K$ 
7:      $K' \leftarrow K \setminus L \cup M$  ▷ the updated partition  $K$ 
8:      $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{K'\}$  ▷ update resulting partitioning
9:   end for
10:   $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{\emptyset\}$  ▷ remove all empty partitions
11:  return  $\mathcal{P}'$ 
12: end procedure

```

Algorithm 1 determines the partitioning that *would* result if a candidate transaction ct was applied to \mathcal{R}_{opt} . That is because the underlying acquaintances graph is not modified. The final application of a selected candidate transaction to the acquaintances graph is explained in Section 8.4. Since a candidate transaction contains only candidate moves that do not depend on each other, they can be performed in an arbitrary order. The procedure ensures that all agents whose target partition is specified as $K_{\text{to}} = \emptyset$ are assigned to the same new partition (cf. Line 4). Partitions that lose all members in the course of a candidate transaction’s application dissolve, i.e., they are removed from the partitioning (cf. Line 10). With regard to our running example, performing the above-given candidate transaction in the context of N ’s regional partitioning to optimize \mathcal{R}_{opt} yields $\mathcal{R}'_{\text{opt}} = \{K \setminus \{a_4\}, L, N \setminus \{a_{13}, a_{14}\} \cup \{a_4\}, \{a_{13}, a_{14}\}\}$.

Because the size of the regional search space decreases exponentially with the cardinality of CM_{in} and K_{opt} , SPADA uses a heuristic providing the leaders with the most $\#CM_{\text{in}}$ and $\#CM_{\text{out}}$ promising candidate moves for the creation of candidate transactions.⁶ To this end, the heuristic utilizes the fitness function f that evaluates the overall partitioning’s quality in the application-specific setting. Here, f is used to assess the change in fitness $\Delta f((a, K_{\text{from}}, K_{\text{to}}), \mathcal{R}_{\text{opt}})$ that would result from the *isolated* application of a candidate move $(a, K_{\text{from}}, K_{\text{to}})$ to the regional partition to optimize \mathcal{R}_{opt} :

$$\Delta f((a, K_{\text{from}}, K_{\text{to}}), \mathcal{R}_{\text{opt}}) = f(\mathcal{R}'_{\text{opt}}) - f(\mathcal{R}_{\text{opt}})$$

$$\text{with } \mathcal{R}'_{\text{opt}} = \mathcal{R}_{\text{opt}} \setminus \{K_{\text{from}}, K_{\text{to}}\} \cup \{K_{\text{from}} \setminus \{a\}, K_{\text{to}} \cup \{a\}\}$$

The greater $\Delta f((a, K_{\text{from}}, K_{\text{to}}), \mathcal{R}_{\text{opt}})$, the more promising the candidate move. Of course, the relevance of the exact value of the change in fitness diminishes when a candidate move is combined with others but still it gives a rough indication whether it might be able to improve the regional partitioning. The indication’s relevance mainly depends on the application-specific fitness function used for optimization. For instance, the locality of the “classic” k-means distance measure – as used in clustering and anticlustering – is likely to cause more sustainable indications than the fitness functions of homogeneous partitioning where the quality of a single partition depends on the composition of all other partitions. Applying the heuristic to CM_{in} and CM_{out} separately ensures that the leaders always keep the ability to incorporate new agents into and to exclude members from their partition.

In order to find a candidate transaction suitable for application, the leader finally uses Algorithm 2 to create $\#CT$ candidate transactions on the basis of the remaining candidate moves CM_{in} and CM_{out} . This algorithm can be thought of as a *random search* where the hypersphere contains the remaining elements of CM_{in} and CM_{out} . All generated candidate transactions are gathered in a set. Having created the candidate transactions, the leader corrects them as described in Section 8.3 so that they adhere to the partitioning constraints, chooses the best one for execution, and applies the changes to the acquaintances graph as explained in Section 8.4.

⁶ $\#CM_{\text{in}}$ and $\#CM_{\text{out}}$ depend on the cardinality of CM_{in} and CM_{out} .

Algorithm 2 Creation of a Candidate Transaction

Require: \mathcal{R}_{opt} is the regional partitioning to optimize, CM_{in} is a proper set of candidate moves for integration, CM_{out} is the corresponding set of candidate moves for exclusion, $p_{CM_{\text{in}}} \in [0, 1]$ parametrizes a Bernoulli distribution used to take candidate moves from CM_{in} with probability $p_{CM_{\text{in}}}$, $\#ITERS_{\text{max}} > 0$ is the maximum number of iterations performed and an upper bound for the number of candidate moves in the generated candidate transaction.

Ensure: ct is a set of not more than $\#ITERS_{\text{max}}$ conflict-free candidate moves whose application would increase the fitness of the regional partitioning to optimize.

```

1: procedure CREATECANDIDATETRANSACTION( $\mathcal{R}_{\text{opt}}, CM_{\text{in}}, CM_{\text{out}}, p_{CM_{\text{in}}}, \#ITERS_{\text{max}}$ )
2:    $ct \leftarrow \emptyset$  ▷ initially empty candidate transaction
3:    $\mathcal{R}'_{\text{opt}} \leftarrow \text{DETERMINERESULTINGPARTITIONING}(\mathcal{R}_{\text{opt}}, ct)$  ▷ resulting regional partitioning
4:    $temp_{\text{in}} \leftarrow \emptyset, temp_{\text{out}} \leftarrow \emptyset$  ▷ helper sets holding candidate moves that decrease the fitness
5:    $iter \leftarrow 0$  ▷ counts the number of iterations
6:
7:   ▷ iteratively create the candidate transaction; terminate if max. no. of iterations reached or no
   candidate moves are left
8:   while  $iter < \#ITERS_{\text{max}} \wedge (CM_{\text{in}} \neq \emptyset \vee CM_{\text{out}} \neq \emptyset)$  do
9:      $CM \leftarrow \text{CHOOSEBERNOULLI}(p_{CM_{\text{in}}}, CM_{\text{in}}, CM_{\text{out}})$  ▷ selects either  $CM_{\text{in}}$  with probability
        $p_{CM_{\text{in}}}$  or  $CM_{\text{out}}$  with probability  $1 - p_{CM_{\text{in}}}$ ; always chooses a non-empty set
10:     $(a, K_{\text{from}}, K_{\text{to}}) \leftarrow \text{CHOOSEUNIFORM}(CM)$  ▷ arbitrary candidate move
11:     $ct' \leftarrow ct \cup \{(a, K_{\text{from}}, K_{\text{to}})\}$  ▷ add candidate move to candidate transaction
12:     $\mathcal{R}''_{\text{opt}} \leftarrow \text{DETERMINERESULTINGPARTITIONING}(\mathcal{R}_{\text{opt}}, ct')$  ▷ alternative resulting regional
       partitioning
13:
14:    ▷ decide whether to keep or discard the candidate move
15:    if  $f(\mathcal{R}''_{\text{opt}}) > f(\mathcal{R}'_{\text{opt}})$  then ▷ only candidate moves increasing the fitness are kept
16:       $\mathcal{R}'_{\text{opt}} \leftarrow \mathcal{R}''_{\text{opt}}$  ▷ update the resulting regional partitioning
17:       $ct \leftarrow ct'$  ▷ update the candidate transaction
18:      ▷ bad candidate moves might have become beneficial
19:       $CM_{\text{in}} \leftarrow CM_{\text{in}} \cup temp_{\text{in}}, CM_{\text{out}} \leftarrow CM_{\text{out}} \cup temp_{\text{out}}$ 
20:       $temp_{\text{in}} \leftarrow \emptyset, temp_{\text{out}} \leftarrow \emptyset$ 
21:      ▷ remove all pending candidate moves involving the agent of  $(a, K_{\text{from}}, K_{\text{to}})$  from  $CM_{\text{out}}$ 
22:       $CM_{\text{confl}} \leftarrow \{(a', K'_{\text{from}}, K'_{\text{to}}) \mid (a', K'_{\text{from}}, K'_{\text{to}}) \in CM_{\text{out}} \wedge a' = a\}$  ▷ conflicting moves
23:       $CM_{\text{out}} \leftarrow CM_{\text{out}} \setminus CM_{\text{confl}}$  ▷ remove conflicting candidate moves
24:      else ▷ remember bad candidate move
25:        if  $CM = CM_{\text{in}}$  then
26:           $temp_{\text{in}} \leftarrow temp_{\text{in}} \cup \{(a, K_{\text{from}}, K_{\text{to}})\}$ 
27:        else
28:           $temp_{\text{out}} \leftarrow temp_{\text{out}} \cup \{(a, K_{\text{from}}, K_{\text{to}})\}$ 
29:        end if
30:      end if
31:
32:       $CM \leftarrow CM \setminus \{(a, K_{\text{from}}, K_{\text{to}})\}$  ▷ candidate move has been processed
33:       $iter \leftarrow iter + 1$ 
34:    end while
35:
36:    return  $ct$ 
37: end procedure

```

Starting with an initially empty candidate transaction ct , Algorithm 2 tries to iteratively extend ct by beneficial and randomly chosen candidate moves. In each iteration, Algorithm 2 chooses to take a candidate move from CM_{in} with probability $p_{CM_{in}} \in [0, 1]$ or from CM_{out} with probability $1 - p_{CM_{in}}$ (cf. Lines 9 and 10). Afterwards, it evaluates whether the inclusion of the selected candidate move *further* increases the fitness of the partitioning that would result if the candidate transaction ct was applied to \mathcal{R}_{opt} (cf. Line 15). This procedure allows SPADA to take advantage of synergy effects. If the fitness continues to increase, the candidate move remains in ct (cf. Line 17) and possibly conflicting candidate moves are removed from CM_{out} (cf. Lines 22 and 23). Otherwise, the candidate move is considered inappropriate, which is why it is added to and stays on a blacklist (cf. Lines 25 to 29) until ct has been extended by another candidate move (cf. Lines 19 and 20). The creation of ct is finished if either a maximum of $\#ITERS_{max}$ iterations has been performed or CM_{in} as well as CM_{out} are empty. The latter is the case either if all candidate moves have been added to the candidate transaction, or if the remaining candidate moves are in conflict or considered to be inappropriate. $\#ITERS_{max}$ is thus an upper bound for the number of candidate moves in ct .

In order to gain a target-oriented optimization, the most important characteristic of Algorithm 2 is that it guarantees the generation of candidate transactions whose application would increase the fitness of \mathcal{R}_{opt} . To achieve a fair balance between computational costs and solution quality, we opted for an iterative search strategy. For the same reason and in order to stimulate exploration and mitigate the attraction of local optima, Algorithm 2 does not take the partitioning constraints into account. The candidate transactions' compliance with the partitioning constraints is ensured in a dedicated correction phase afterwards. Because the correction of a candidate transaction can have unexpected effects on the way its application would change the fitness of \mathcal{R}_{opt} , the leader creates multiple candidate transactions. That way, it can choose the most profitable transaction subsequent to the correction phase, which alleviates the unexpected effects the corrections might cause.

Shuffling Acquaintances

Having optimized its regional partitioning, a leader shuffles the acquaintances of its partition members so that they become acquainted with various different agents over time. Because agents keep their acquaintances whenever they change partitions, this measure increases the variety in regional partitionings the leaders regard during the optimization. Due to the broad spectrum of agents and partitions becoming available as potential members and sources or targets for candidate moves, as well as the different perspectives from which the leaders assess (regions of) the overall partitioning, SPADA makes better use of regional knowledge and avoids local optima. Together with the iterative refinement of overlapping partial solutions, this principle ultimately allows SPADA to form a suitable overall partitioning.

When shuffling acquaintances, a leader modifies only the mapping between its members and their acquaintances, that is, it shuffles the heads of the unmarked edges of its partition. This entails that (1) the direction of unmarked edges is not changed and (2) that the partition's marked edges and thus the structure of its arborescence are not altered. For example, two unmarked edges (a_i, a_j) and (a_k, a_l) can either be replaced by (a_i, a_l) and (a_k, a_j) or kept unchanged. In any case, shuffling the acquaintances of the members of a partition K_{opt} does not change their accumulated set of acquaintances $\{acqAgents(a) \mid a \in K_{opt}\}$. So if a member of K_{opt} has been acquainted with an agent a_m , shuffling acquaintances ensures that at least one of the members of K_{opt} will be acquainted with a_m . Because all modifications also comply with the other properties of a proper acquaintances graph (cf. Section 8.1), its weak connectivity is maintained.

8.3 Satisfaction of Partitioning Constraints

To enable the formation of scalable hierarchical system structures, SPADA supports compliance with the partitioning constraints introduced in Section 6.2. Because leaders disregard the partitioning constraints during the creation of candidate transactions for the sake of better exploration and reduced complexity, they correct their created candidate transactions using the technique described at the end of this section.

Applying a repaired candidate transaction to the regional partitioning to optimize \mathcal{R}_{opt} yields an overall partitioning that is guaranteed to adhere to the minimum- and maximum-partition-size as well as the minimum- and maximum-number-of-partitions constraints. Of course, this is only one of several possible ways to deal with infeasible candidate transactions: Instead of correcting all generated candidate transactions, leaders could also degrade the fitness of infeasible candidate transactions, which requires penalties in line with the degree of constraint violation. If the search space is not restricted to the solution space, the chosen candidate transaction does not necessarily have to be corrected (cf. [52]). In this case, the overall partitioning must be repaired right before the algorithm terminates in order to return a feasible solution. These and multiple other constraint handling techniques for heuristics are discussed in [145]. Since the correction of a candidate solution is likely to change its quality, the drawback of allowing infeasible interim solutions is that the solution's actual quality remains unclear until the algorithm terminates. Furthermore, the evaluation of the PSOPP algorithm showed that the mere size of the search space favors its restriction to feasible solutions (see Section 7.6). We outline and justify our approach in the remainder of this section.

While the partition-size constraints are local constraints whose satisfaction can be enforced by each leader individually on the basis of local information, leaders require global information – namely the total number of partitions – to make sure that the application of a candidate transaction would satisfy the number-of-partitions constraints. As we will learn in the following subsection, this global information is actually only required once for the first correction of the overall partitioning. In all following optimization cycles, the partitions need only local information to satisfy the constraints.

Localization of the Global Number-of-Partitions Constraints

The first correction of the number of partitions has to be performed by a designated leader for the overall partitioning, i.e., in a centralized manner. It can be performed either during SPADA's initialization so that the number-of-partitions constraints are satisfied in all optimization steps, or after a predefined optimization cycle. In the latter case, the leaders ignore the number-of-partitions constraints until the centralized correction has been performed. Preliminary results showed that the accompanying increase in SPADA's exploration can turn out to be very beneficial in the context of highly constrained search spaces that cause sparse solution spaces. In the exploration phase in which SPADA forms a partitioning \mathcal{P} that does not necessarily satisfy $n_{\min} \leq |\mathcal{P}| \leq n_{\max}$, SPADA can search for a promising region in the search space. The one-time centralized correction then transforms the possibly infeasible assignment of decision variables into a feasible assignment, which is improved in all following optimization steps without leaving the solution space. Note that the satisfaction of the partition-size constraints is always ensured since only local knowledge is required.

The designated leader performs the one-time centralized correction of the number of partitions in a partitioning \mathcal{P} by means of Algorithm 3. This algorithm makes minimally invasive changes in order to satisfy the partitioning constraints but pays no attention to how these changes influence the partitioning's fitness. The necessary changes are recorded in an initially empty candidate transaction ct and adopted by its subsequent application to \mathcal{P} . The resulting partitioning \mathcal{P}' satisfies all partitioning constraints. For this purpose, Algorithm 3 requires that all partitions $K \in \mathcal{P}$ satisfy $s_{\min} \leq |K| \leq s_{\max}$. As stated above, this is ensured by all leaders locally. Of course, \mathcal{P} can only be corrected if the underlying set of agents \mathfrak{A} meet Equation (6.3) (this is a prerequisite for running SPADA, though). The algorithm is sound and complete, i.e., it successfully repairs every partitioning that meets these conditions. In Algorithm 3, Δn denotes the number of partitions that have to be dissolved (if $\Delta n < 0$) or created (if $\Delta n > 0$) so that $n_{\min} \leq |\mathcal{P}'| \leq n_{\max}$ holds. Lines 4 to 22 and Lines 24 to 43 ensure that \mathcal{P}' does not exceed n_{\max} and does not fall below n_{\min} , respectively. If there are too many partitions, Algorithm 3 dissolves the $-\Delta n = |\mathcal{P}'| - n_{\max}$ smallest partitions by iteratively moving their members to the next smallest partition in \mathcal{P}' . Otherwise if there are too few partitions, the algorithm creates $\Delta n = n_{\min} - |\mathcal{P}'|$ new partitions of the smallest possible size s_{\min} by iteratively removing an agent from the largest partition in \mathcal{P}' . In this process, Lines 13 to 17 and 33 to 37 ensure that the candidate transaction ct remains conflict-free despite its iterative generation.

Algorithm 3 Correction of the Number of Partitions

Require: ct is a (conflict-free) candidate transaction whose application to \mathcal{P} yields partitions of correct size, \mathcal{P} is a partitioning that satisfies the minimum- and maximum-partition-size constraints, Δn denotes the number of partitions that have to be dissolved (if $\Delta n < 0$) or created (if $\Delta n > 0$), and s_{min} is the minimum size of partitions.

Ensure: ct is a candidate transaction whose application to \mathcal{P} creates a partitioning that satisfies all partitioning constraints.

```
1: procedure REPAIRNUMBEROFPARTITIONS( $ct, \mathcal{P}, \Delta n, s_{min}$ )
2:    $\mathcal{P}' \leftarrow \text{DETERMINERESULTINGPARTITIONING}(\mathcal{P}, ct)$  ▷ resulting partitioning
3:
4:   while  $\Delta n < 0$  do ▷ too many partitions:  $\max\{0, -\Delta n\}$  partitions have to be merged
5:      $K \leftarrow \text{SMALLEST}(\mathcal{P}')$  ▷ smallest partition from the resulting partitioning
6:      $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{K\}$ 
7:     for all  $a \in K$  do ▷ dissolve the smallest partition
8:        $K_{to} \leftarrow \text{SMALLEST}(\mathcal{P}')$  ▷ next smallest partition is target for move
9:        $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{K_{to}\} \cup \{K_{to} \cup \{a\}\}$  ▷ adjust  $\mathcal{P}'$  by adding  $a$  to  $K_{to}$ 
10:       $K_{from} \leftarrow K$ 
11:
12:      ▷ prevent conflicting candidate moves: “redirect” conflicting move to  $K_{to}$ 
13:      if  $\text{CONSTRAINS\CANDIDATEMOVEWITHAGENT}(ct, a)$  then
14:         $(a, K'_{from}, K) \leftarrow \text{GETCANDIDATEMOVEWITHAGENT}(ct, a)$ 
15:         $ct \leftarrow ct \setminus \{(a, K'_{from}, K)\}$ 
16:         $K_{from} \leftarrow K'_{from}$  ▷ instead of moving  $a$  to  $K$  it is now moved to  $K_{to}$ 
17:      end if
18:
19:       $ct \leftarrow ct \cup \{(a, K_{from}, K_{to})\}$  ▷ add new candidate move to candidate transaction
20:    end for
21:     $\Delta n \leftarrow \Delta n + 1$ 
22:  end while
23:
24:  while  $\Delta n > 0$  do ▷ too few partitions:  $\max\{0, \Delta n\}$  partitions have to be created
25:     $K_{to} \leftarrow \emptyset$  ▷ new initially empty partition
26:    while  $|K_{to}| < s_{min}$  do ▷ create new partition of minimal size
27:       $K \leftarrow \text{LARGEST}(\mathcal{P}')$  ▷ remove member from the largest partition
28:       $a \leftarrow \text{CHOOSEUNIFORM}(K)$  ▷ arbitrary agent from  $K$ 
29:       $\mathcal{P}' \leftarrow \mathcal{P}' \setminus \{K\} \cup \{K \setminus \{a\}\}$  ▷ adjust  $\mathcal{P}'$  by removing  $a$  from  $K$ 
30:       $K_{from} \leftarrow K$ 
31:
32:      ▷ prevent conflicting candidate moves: “redirect” conflicting move to  $K_{to}$ 
33:      if  $\text{CONSTRAINS\CANDIDATEMOVEWITHAGENT}(ct, a)$  then
34:         $(a, K'_{from}, K) \leftarrow \text{GETCANDIDATEMOVEWITHAGENT}(ct, a)$ 
35:         $ct \leftarrow ct \setminus \{(a, K'_{from}, K)\}$ 
36:         $K_{from} \leftarrow K'_{from}$  ▷ instead of moving  $a$  to  $K$  it is now moved to  $K_{to}$ 
37:      end if
38:
39:       $ct \leftarrow ct \cup \{(a, K_{from}, K_{to})\}$  ▷ add new candidate move to candidate transaction
40:    end while
41:     $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{K_{to}\}$  ▷ add new partition to resulting partitioning
42:     $\Delta n \leftarrow \Delta n - 1$ 
43:  end while
44: end procedure
```

Subsequent to the one-time centralized correction of \mathcal{P} yielding \mathcal{P}' , the designated leader localizes the global number-of-partitions constraints by decomposing them into several local counterparts. The localization is achieved by distributing the remaining degrees of freedom in terms of how many partitions may be created or dissolved without exceeding n_{max} or falling below n_{min} to the partitions in \mathcal{P}' . To this end, *partition-specific boundaries* $\Delta n_{new}(K)$ and $\Delta n_{dis}(K)$ state how many partitions a partition $K \in \mathcal{P}'$ contributes for creation and dissolution the next time it participates in a regional optimization. For example, a parametrization of $\Delta n_{new}(K) = 2$ and $\Delta n_{dis}(K) = 3$ means that K contributes the opportunity of creating up to two and dissolving up to three partitions. Overall, $n_{max} - |\mathcal{P}'|$ partitions may be created and $|\mathcal{P}'| - n_{min}$ partitions may be dissolved after the centralized correction without violating the number-of-partitions constraints in the upcoming optimization cycle. The designated leader distributes these degrees of freedom entirely (that is, $n_{max} - |\mathcal{P}'| = \sum_{K \in \mathcal{P}'} \Delta n_{new}(K)$ and $|\mathcal{P}'| - n_{min} = \sum_{K \in \mathcal{P}'} \Delta n_{dis}(K)$), but as evenly as possible, to the partitions in \mathcal{P}' . The even distribution of the remaining degrees of freedom avoids their unintended concentration in specific partitions. This idea of localizing the global number-of-partitions constraints is inspired by the decomposition of global quality of service constraints for the distributed composition of web services by Alrifai and Risse [5]. However, the nature of the quality of service constraints and the problem of service composition require to solve a dedicated optimization problem to determine a proper decomposition. This is, thankfully, not needed in case of our global number-of-partitions constraints.

Example – Initial Partition-Specific Boundaries for Creating and Dissolving Partitions

Let us regard a partitioning \mathcal{Q} consisting of three partitions K, L , and M . Assume that the number-of-partitions constraints are parametrized with $n_{min} = 1$ and $n_{max} = 6$. For \mathcal{Q} in its entirety, up to $\Delta n_{new}(\mathcal{Q}) = 6 - 3 = 3$ partitions may be created and no more than $\Delta n_{dis}(\mathcal{Q}) = 3 - 1 = 2$ partitions may be dissolved. In this setting, the following partition-specific boundaries conform with the even distribution of the degrees of freedom: $\Delta n_{new}(K) = \Delta n_{new}(L) = \Delta n_{new}(M) = 1$, $\Delta n_{dis}(K) = \Delta n_{dis}(L) = 1$, and $\Delta n_{dis}(M) = 0$.

When carrying out a regional optimization, the local correction of candidate transactions explained at the end of this section ensures compliance with these boundaries. To revise the composition of a regional partitioning to optimize \mathcal{R}_{opt} , the leader $l(K_{opt})$ performing the optimization aggregates the degrees of freedom of all partitions in \mathcal{R}_{opt} . Hence, $l(K_{opt})$ is permitted to create up to $\Delta n_{new}(\mathcal{R}_{opt}) = \sum_{K \in \mathcal{R}_{opt}} \Delta n_{new}(K)$ and dissolve up to $\Delta n_{dis}(\mathcal{R}_{opt}) = \sum_{K \in \mathcal{R}_{opt}} \Delta n_{dis}(K)$ partitions for this regional optimization.

Definition 10 (Localized Number-of-Partitions Constraints). *When correcting candidate transactions during the optimization of a regional partitioning \mathcal{R}_{opt} , the following **localized number-of-partitions constraints** replace the global number-of-partitions constraints defined in Equation (6.2):*

$$\Delta n_{dis}(\mathcal{R}_{opt}) \leq |\mathcal{R}'_{opt}| - |\mathcal{R}_{opt}| \leq \Delta n_{new}(\mathcal{R}_{opt}) \quad (8.1)$$

\mathcal{R}'_{opt} denotes the regional partitioning that would result from the application of a corrected candidate transaction to \mathcal{R}_{opt} . The boundaries $\Delta n_{dis}(\mathcal{R}_{opt})$ and $\Delta n_{new}(\mathcal{R}_{opt})$ are the upper limits for creating and dissolving partitions in the regional optimization.

After applying a corrected candidate transaction to \mathcal{R}_{opt} , which yields the partitioning \mathcal{R}'_{opt} , $l(K_{opt})$ has to update the boundaries for all partitions in \mathcal{R}'_{opt} . The remaining degrees of freedom depend on the number of partitions that have been created or dissolved. To be exact, the new boundaries for creating and dissolving partitions in \mathcal{R}'_{opt} are $\Delta n_{new}(\mathcal{R}'_{opt}) = \Delta n_{new}(\mathcal{R}_{opt}) - (|\mathcal{R}'_{opt}| - |\mathcal{R}_{opt}|)$ and $\Delta n_{dis}(\mathcal{R}'_{opt}) = \Delta n_{dis}(\mathcal{R}_{opt}) + (|\mathcal{R}'_{opt}| - |\mathcal{R}_{opt}|)$. These degrees of freedom are distributed the same way as explained above, but this time only to the partitions in \mathcal{R}'_{opt} . Because all regional optimizations adhere to and update these boundaries as described, the overall partitioning satisfies the number-of-partitions constraints.

Example – Regional Update of the Partition-Specific Boundaries

Regarding the setting in the above-given example, assume that the leader of partition K revises the regional partitioning to optimize $\mathcal{R}_{\text{opt}} = \{K, M\}$. In this region, the leader is permitted to create $\Delta n_{\text{new}}(\mathcal{R}_{\text{opt}}) = \Delta n_{\text{new}}(K) + \Delta n_{\text{new}}(M) = 1 + 1 = 2$ and dissolve $\Delta n_{\text{dis}}(\mathcal{R}_{\text{opt}}) = \Delta n_{\text{dis}}(K) + \Delta n_{\text{dis}}(M) = 1 + 0 = 1$ partitions. Let us assume that it creates an additional partition N resulting in the updated regional partitioning $\mathcal{R}'_{\text{opt}} = \{K', M', N\}$ and in the overall partitioning $\mathcal{Q}' = \{K', L, M', N\}$. For $\mathcal{R}'_{\text{opt}}$, $\Delta n_{\text{new}}(\mathcal{R}'_{\text{opt}}) = 2 - (3 - 2) = 1$ and $\Delta n_{\text{dis}}(\mathcal{R}'_{\text{opt}}) = 1 + (3 - 2) = 2$ are the remaining degrees of freedom. The following partition-specific boundaries can result from the regional update for the involved partitions K', M', N : $\Delta n_{\text{new}}(K') = 1$, $\Delta n_{\text{new}}(M') = \Delta n_{\text{new}}(N) = 0$, $\Delta n_{\text{dis}}(K') = \Delta n_{\text{dis}}(M') = 1$, and $\Delta n_{\text{dis}}(N) = 0$. Because L is not involved in the regional optimization, it keeps its boundaries $\Delta n_{\text{new}}(L) = \Delta n_{\text{dis}}(L) = 1$. Consequently, the overall resulting partitioning \mathcal{Q}' is now allowed to create and dissolve a total of $\Delta n_{\text{new}}(\mathcal{Q}') = 1 + 1 + 0 + 0 = 2$ and $\Delta n_{\text{dis}}(\mathcal{Q}') = 1 + 1 + 1 + 0 = 3$ partitions, compared to $\Delta n_{\text{new}}(\mathcal{Q}) = 3$ and $\Delta n_{\text{dis}}(\mathcal{Q}) = 2$ before the creation of partition N .

Local Correction of Candidate Transactions

In order to abide by the partitioning constraints, each leader corrects its generated candidate transactions locally before it chooses the best transaction increasing the regional fitness for application. The correction of candidate transactions is a four-stage process, illustrated in Figure 8.6: (1) The first stage utilizes remaining candidate moves to repair the candidate transactions by extending them in a target-oriented manner. In this stage, no guarantees of success can be given. (2) Afterwards, the second correction stage ensures that all candidate transactions would lead to a partitioning that satisfies the partition-size constraints. This is achieved by removing a sufficient number of candidate moves from each candidate transaction. (3) By merging existing or creating new partitions, the number-of-partitions constraints hold after the third stage. This stage is excluded until the one-time centralized correction has been performed. (4) The last stage serves as a filter making sure that no candidate transaction can be selected for application that does not increase the fitness of the regional partitioning to optimize. As stated in Section 8.2, the correction of a candidate transaction might have unexpected effects on its valuation, which is why it has to be re-evaluated after the third stage.

In the following paragraphs, we explain the first three correction stages in more detail. The fourth and last stage simply removes all candidate transactions that do not increase the fitness. Given that the current partitioning satisfies the partitioning constraints, the presented approach is able to correct every candidate transaction. So, no matter which transaction is ultimately chosen for application, the resulting partitioning is valid.

Stage 1 – Extension of Candidate Transactions Before correcting the size and the number of the resulting partitions in a rather rigorous manner, the leader $l(K_{\text{opt}})$ performing the optimization of the region \mathcal{R}_{opt} tries to improve the correctness of the candidate transactions by making use of the remaining conflict-free candidate moves for integration CM_{in} and exclusion CM_{out} . For each too large partition K in the resulting region $\mathcal{R}'_{\text{opt}}$, $l(K_{\text{opt}})$ checks if there is a sufficient number of conflict-free candidate moves that, if added to the candidate transaction, would decrease the size of K to the allowed maximum s_{max} . If this is the case, $l(K_{\text{opt}})$ adds $|K| - s_{\text{max}}$ of these candidate moves to the candidate transaction. Further, for each too small partition $L \in \mathcal{R}'_{\text{opt}}$, $l(K_{\text{opt}})$ decides whether the remaining conflict-free candidate moves should be used to increase L 's size so that $|L| = s_{\text{min}}$ or to dissolve it. The latter goal is only pursued if the resulting partitioning would be too large (i.e., if $|\mathcal{R}'_{\text{opt}}| - |\mathcal{R}_{\text{opt}}| > \Delta n_{\text{new}}(\mathcal{R}_{\text{opt}})$). In all other cases, the former goal is pursued, which improves SPADA's ability to create new, sufficiently large partitions. As is the case with too large partitions, too small partitions are only corrected if a sufficient number of helpful candidate moves exist. All helpful candidate moves are added to the corresponding candidate transaction.

While this stage is not mandatory for correcting the candidate transactions, it promotes exploration and increases the probability of creating new, sufficiently large partitions. The resulting candidate

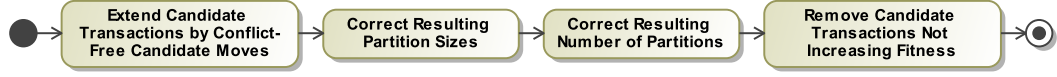


Figure 8.6: Overview of the four stages to correct candidate transactions.

transactions do not necessarily lead to valid partitionings. This property is obtained by the following two correction stages.

Stage 2 – Correction of Resulting Partition Sizes The second correction stage repairs the candidate transactions with regard to the size of the resulting partitions. Instead of extending the candidate transactions, it selectively removes specific candidate moves from them in an iterative manner. It guarantees that every candidate transaction can be repaired and that the corrected candidate transactions comply with the partition-size constraints. This is because the current partitioning satisfies the partitioning constraints, and, in the worst case, all candidate moves are removed from a candidate transaction so that no changes would be made. The complete procedure for correcting the partition sizes of a specific candidate transaction is shown in Algorithm 4. It is performed once for each generated candidate transaction. When correcting a candidate transaction ct , the leader $l(K_{\text{opt}})$ considers the two cases of too large and too small partitions. In the former case, $l(K_{\text{opt}})$ removes $|K'| - s_{\text{max}}$ candidate moves that would transfer an agent into the too large partition K' (cf. Lines 10 to 15). Hence, only as many agents as possible are moved into K' . In the latter case, $l(K_{\text{opt}})$ distinguishes between already existing partitions and the new partition that can be created by $l(K_{\text{opt}})$. If the partition already exists, $l(K_{\text{opt}})$ removes $s_{\text{min}} - |K'|$ candidate moves that would take an agent out of K' (cf. Lines 17 to 22). In case of the new but too small partition, $l(K_{\text{opt}})$ removes all $|K'|$ candidate moves causing the partition's creation (cf. Lines 10 to 15), which highlights the benefit of the first correction stage.

Stage 3 – Correction of the Resulting Number of Partitions Having corrected the candidate transaction with regard to the partition-size constraints, the leader $l(K_{\text{opt}})$ makes sure that, in the end, all partitioning constraints are satisfied. To this end, it utilizes Algorithm 3 – the same algorithm SPADA employs for the one-time centralized correction of the resulting number of partitions. But this time Algorithm 3 is called on the basis of regional instead of global knowledge. The number of partitions Δn that have to be dissolved or created stems from the resulting regional partitioning to optimize $\mathcal{R}'_{\text{opt}}$ and the regional boundaries indicating how many partitions may be created $\Delta n_{\text{new}}(\mathcal{R}_{\text{opt}})$ or dissolved $\Delta n_{\text{dis}}(\mathcal{R}_{\text{opt}})$. In case there would be too many partitions, $l(K_{\text{opt}})$ dissolves $-\Delta n = -(\Delta n_{\text{new}}(\mathcal{R}_{\text{opt}}) - (|\mathcal{R}'_{\text{opt}}| - |\mathcal{R}_{\text{opt}}|))$ partitions. If there would be too few partitions, $l(K_{\text{opt}})$ creates $\Delta n = -(\Delta n_{\text{dis}}(\mathcal{R}_{\text{opt}}) + (|\mathcal{R}'_{\text{opt}}| - |\mathcal{R}_{\text{opt}}|))$ partitions. The accompanying modifications of the candidate transaction have been described at the beginning of this section. Afterwards, the partition-size as well as the localized number-of-partitions constraints (cf. Equation (8.1)) hold for each candidate transaction.

Note that the correction of the number of partitions is only possible by permitting the leader to create and add candidate moves that affect arbitrary agents in \mathcal{R}_{opt} and have arbitrary sources and targets in \mathcal{R}_{opt} . Consequently, these additional candidate moves are not necessarily contained in the sets of candidate moves for integration CM_{in} or exclusion CM_{out} . Recalling that the main purpose of restricting CM_{in} and CM_{out} was to limit the size of the search space (cf. Section 8.2), this restriction is not needed at this point: Due to the selective changes made to the candidate transactions, the size of the search space is irrelevant. On the other hand, the restriction has to be omitted in this correction stage to be able to repair arbitrary candidate transactions.

8.4 Applying Partition Changes to the Acquaintances Graph

Having chosen a suitable candidate transaction, the regarded leader $l(K_{\text{opt}})$ sequentially carries out the contained candidate moves. Afterwards, the acquaintances graph reflects the revised partitioning as given by Algorithm 1. As mentioned before, the order in which the moves are applied is irrelevant.

Algorithm 4 Correction of the Partitions' Size

Require: ct is a (conflict-free) candidate transaction, \mathcal{R}_{opt} is a partitioning that satisfies all partitioning constraints, $1 \leq s_{\min} \leq s_{\max}$ holds for the minimum s_{\min} and maximum s_{\max} size of partitions.

Ensure: ct' is a candidate transaction whose resulting partitioning consists of partitions of correct size.

```

1: procedure REPAIRSIZEOFPARTITIONS( $ct, \mathcal{R}_{\text{opt}}, s_{\min}, s_{\max}$ )
2:    $ct' \leftarrow ct$  ▷ corrected candidate transaction
3:   ▷ correct the size of each resulting partition  $K'$ 
4:   for all  $K \in \mathcal{P} \cup \{\emptyset\}$  do ▷  $\emptyset$  represents the possibly newly created partition
5:      $L \leftarrow \{a \mid (a, K_{\text{from}}, K_{\text{to}}) \in ct \wedge K_{\text{from}} = K\}$  ▷ all agents that are removed from  $K$ 
6:      $M \leftarrow \{a \mid (a, K_{\text{from}}, K_{\text{to}}) \in ct \wedge K_{\text{to}} = K\}$  ▷ all agents that are transferred into  $K$ 
7:      $K' \leftarrow K \setminus L \cup M$  ▷ the resulting partition  $K'$ 
8:     if  $K' \neq \emptyset$  then ▷ process only non-empty resulting partitions
9:       ▷ reduce size of too large partitions or dissolve new partitions of insufficient size
10:      while  $|K'| > s_{\max} \vee (K = \emptyset \wedge 0 < |K'| < s_{\min})$  do
11:        ▷ remove candidate move that increases the partition's size
12:         $(a, K_{\text{from}}, K) \leftarrow \text{CHOOSEUNIFORM}(M)$ 
13:         $ct' \leftarrow ct' \setminus \{(a, K_{\text{from}}, K)\}$ 
14:         $K' \leftarrow K' \setminus \{a\}$ 
15:      end while
16:      ▷ increase size of too small partitions
17:      while  $|K'| < s_{\min}$  do
18:        ▷ remove candidate move that decreases the partition's size
19:         $(a, K, K_{\text{to}}) \leftarrow \text{CHOOSEUNIFORM}(L)$ 
20:         $ct' \leftarrow ct' \setminus \{(a, K, K_{\text{to}})\}$ 
21:         $K' \leftarrow K' \cup \{a\}$ 
22:      end while
23:    end if
24:  end for
25:  return  $ct'$ 
26: end procedure

```

Although the leader makes sequential changes, the transaction appears to be carried out atomically since partitions outside \mathcal{R}_{opt} do not interfere with this process. With respect to the acquaintances graph, moving an agent a_i between two existing partitions requires removing the agent from its current and integrating it into its new arborescence. Both actions are performed by the leaders of the affected partitions. If the new partition of a_i does not yet exist (i.e., if the processed candidate move is the first with target $K_{\text{to}} = \emptyset$), it is sufficient to remove a_i from its current partition. Subsequently, a_i becomes the leader $l(K_{\text{new}}) = a_i$ of the new partition $K_{\text{new}} = \{a_i\}$. Note that K_{new} is a singleton until further agents are added.

In the two following subsections, we describe the removal and integration of agents in detail and show that all modifications comply with the properties of a proper acquaintances graph. These comprise, among others, its weak connectivity, the predefined and constant number of outgoing edges per agent, and the arborescence property of partitions (see Section 8.1).

Removing Agents from the Partitions' Arborescence

If the agent to be removed is the only member of its partition, it is not necessary to explicitly remove it from its arborescence. Instead, it is sufficient to integrate the agent into its new partition as described in the following subsection. That is because the arborescence's set of marked edges is empty. Otherwise, when removing an agent a_i from a partition K_{from} with $|K_{\text{from}}| > 1$, the number of modifications needed depends on the structure of the acquaintances graph and the position of a_i in K_{from} 's arborescence.

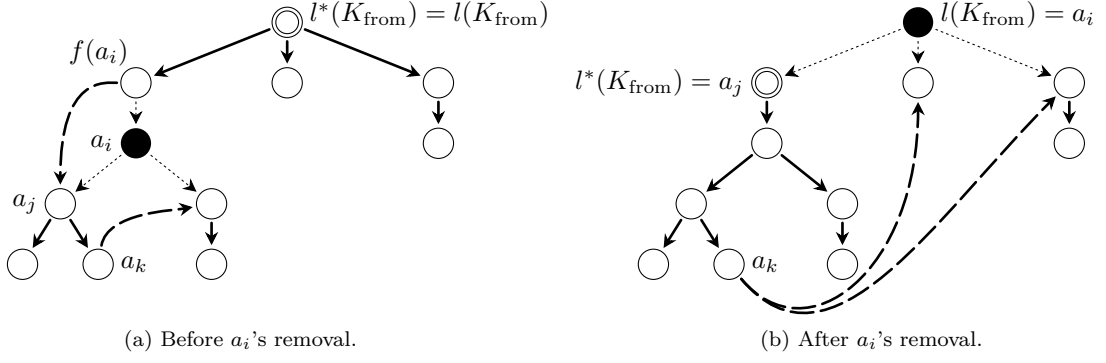


Figure 8.7: The changes made by a leader $l(K_{\text{from}})$ to detach an agent a_i from the arborescence representing its partition: Figure 8.7a depicts the removal of an internal node that is not the root. Figure 8.7b shows the removal of the root, i.e., the leader $l(K_{\text{from}})$. In this case, $l(K_{\text{from}})$ appoints one of its children to be the new leader $l^*(K_{\text{from}})$. In both cases, the root of the resulting arborescence is indicated by a double circle. Solid bold arcs symbolize existing marked edges, dashed bold arcs symbolize new marked edges, and dashed non-bold arcs symbolize marked edges that are converted into unmarked edges.

Basically, K_{from} 's leader removes a_i by converting all of its incoming and outgoing marked edges into unmarked edges by deleting the flag. Afterwards, the leader re-establishes the arborescence property of the remaining partition $K_{\text{from}} \setminus \{a_i\}$. As any agent can be removed from a partition, a_i can be a leaf or an internal node, including the root. If a_i is the root, it is K_{from} 's leader. In such a situation, the leader removes itself from K_{from} and determines a new leader from the set of remaining partition members $K_{\text{from}} \setminus \{a_i\}$. In the following, we describe the necessary modifications of the acquaintances graph for both cases.

If a_i is a leaf, it can be simply removed by deleting the flag from its incoming marked edge. No further modifications are necessary. Otherwise if a_i is an internal node, let $\mathcal{C}(a_i)$ be the set of its children, $f(a_i)$ its parent, and $l^*(K_{\text{from}})$ the leader after a_i has been removed. Furthermore, let \mathcal{M}_{rem} and \mathcal{M}_{new} be two preliminary empty sets of edges. \mathcal{M}_{rem} gathers all marked edges to be converted into unmarked edges, and \mathcal{M}_{new} all new marked edges that might replace existing unmarked edges.

If a_i is an internal node but not the arborescence's root, $l(K_{\text{from}})$ remains the partition's leader (i.e., $l^*(K_{\text{from}}) = l(K_{\text{from}})$) and starts with a_i 's removal by adding a_i 's incoming marked edge $(f(a_i), a_i)$ to \mathcal{M}_{rem} (see Figure 8.7a). In any case, $l(K_{\text{from}})$ adds all outgoing marked edges of a_i to \mathcal{M}_{rem} , i.e., those to its children $\mathcal{C}(a_i)$, in order to separate it from the arborescence. Afterwards, $l(K_{\text{from}})$ selects one of a_i 's children $a_j \in \mathcal{C}(a_i)$. If a_i is not the root, $l(K_{\text{from}})$ creates a new marked edge $(f(a_i), a_j)$ and adds this edge to \mathcal{M}_{new} . This is necessary to prevent the arborescence from falling apart. If a_i is the root, a_j becomes the new root and thus the partition's new leader $l^*(K_{\text{from}}) = a_j$ (see Figure 8.7b). To ensure that all of a_i 's children as well as their descendants remain in the arborescence, $l(K_{\text{from}})$ selects a leaf a_k of the subtree with root a_j (note that this could be a_j itself) and adds a new marked edge (a_k, a_i) for each child $a_l \in \mathcal{C}(a_i) \setminus \{a_j\}$ to \mathcal{M}_{new} . Except for a_j , a_k thus becomes the new parent of each child of a_i .

Leader $l(K_{\text{from}})$ updates the agents' outgoing marked edges by adding all edges contained in \mathcal{M}_{new} ; existing unmarked edges that equal the new marked edges in terms of endpoints are replaced. Subsequently, $l(K_{\text{from}})$ removes the mark from all edges contained in \mathcal{M}_{rem} so that they become unmarked edges. These modifications can lead to a situation in which $f(a_i)$ or a_k exceed the predefined number of outgoing edges by m . This can be the case because $l(K_{\text{from}})$ created a new marked edge for $f(a_i)$ or $|\mathcal{C}(a_i) \setminus \{a_j\}|$ new marked edges for a_k (see Figure 8.7a). As these marked edges do not necessarily replace existing unmarked edges, we have $m \leq 1$ or $m \leq |\mathcal{C}(a_i) \setminus \{a_j\}|$, respectively. If $m > 0$, $l(K_{\text{from}})$ removes m unmarked edges for the corresponding agent in order to remedy the deficiency. When removing these edges, the acquaintances graph stays weakly connected. This can be ensured because, in case of $f(a_i)$, we can simply remove the unmarked edge $(f(a_i), a_i)$ since $f(a_i)$ is acquainted

with a_j via a marked edge, and a_i with a_j via an unmarked edge. With regard to the former leaf a_k , we guarantee the weak connectivity by making sure that a_i is acquainted with m heads of a_k 's unmarked edges so that these edges can be deleted (note that a_i might already be acquainted with some of these agents). Each unmarked edge that has to be newly created for a_i replaces one of a_i 's unmarked edges pointing to its former children $\mathcal{C}(a_i) \setminus \{a_j\}$ that are now the children of a_k . The unmarked edge from a_i to a_j must not be removed to guarantee the graph's weak connectivity.

Having updated the edges, $l(K_{\text{from}})$ informs a_i that it has been removed from K_{from} . In case the leader removed itself from the partition, it informs the remaining partition members about the new leader $l^*(K_{\text{from}})$. As a result, a_i becomes the leader of a new partition $M = \{a_i\}$. If a_i should be moved into another existing partition, $l(K_{\text{from}})$ informs K_{to} 's leader $l(K_{\text{to}})$ that a_i is ready for its integration.

Integrating Agents into the Arborescence of Existing Partitions

When integrating an agent a_i into its new (existing) partition K_{to} , we may assume that it is currently a member of a singleton $K_{\text{dis}} = \{a_i\}$: Either a_i has already been a member of K_{dis} before applying this move so that $K_{\text{dis}} = K_{\text{from}}$, or K_{dis} is the result of removing a_i from its former partition K_{from} . After a_i 's integration, partition K_{dis} dissolves and a_i loses its leader status. For a_i 's integration into K_{to} , we have to distinguish the two following situations:

1. **K_{to} is acquainted with a_i :** Since there is at least one unmarked edge (a_j, a_i) pointing from a member $a_j \in K_{\text{to}}$ to a_i , leader $l(K_{\text{to}})$ integrates a_i into K_{to} simply by adding its flag to (a_j, a_i) . Hence, this unmarked edge is converted into a marked edge. Obviously, this does not break the acquaintances graph's weak connectivity.
2. **K_{to} is not acquainted with a_i :** Such a situation can arise because a leader optimizes its regional partitioning (that encompasses an entire region of the overall acquaintances graph) by creating candidate transactions without taking account of whether or not a member of K_{to} is acquainted with a_i .

Since there is no unmarked edge between K_{to} and a_i , $l(K_{\text{to}})$ has to create a new unmarked edge (a_j, a_i) originating from one of its members $a_j \in K_{\text{to}}$. As the creation of this edge requires $l(K_{\text{to}})$ to delete one of a_j 's unmarked edges, we have to make sure that the acquaintances graph remains weakly connected. In this case, we take advantage of the fact that the partition K_{opt} of the leader $l(K_{\text{opt}})$ that created the candidate move is acquainted with K_{to} , and had been acquainted with K_{from} before executing the transaction. Depending on which other modifications have already been made, K_{opt} 's acquaintance with K_{from} might still be present. To ensure the graph's weak connectivity, we have to consider three different cases:

- a) If a_j holds an unmarked edge into its own partition K_{to} , this edge can be replaced by the required unmarked edge (a_j, a_i) . The weak connectivity is maintained because of the marked edges constituting K_{to} (i.e., there is still an edge that connects a_j with K_{to}).
- b) Otherwise if a_j holds an unmarked edge into K_{opt} , this edge can be replaced by the required unmarked edge (a_j, a_i) . The weak connectivity is maintained because K_{opt} is still acquainted with K_{to} .
- c) Otherwise, a_j holds an unmarked edge (a_j, a_m) pointing to an agent a_m of a further partition N , with $N \neq K_{\text{dis}} \wedge N \neq K_{\text{to}} \wedge N \neq K_{\text{opt}}$. Let us assume that K_{opt} 's acquaintance with K_{to} is realized by the unmarked edge (a_k, a_l) , with $a_k \in K_{\text{opt}}$ and $a_l \in K_{\text{to}}$. In this situation, $l(K_{\text{to}})$ replaces (a_j, a_m) by the required unmarked edge (a_j, a_i) and leader $l(K_{\text{opt}})$ replaces (a_k, a_l) by a new unmarked edge (a_k, a_m) .

If $K_{\text{dis}} = K_{\text{from}}$ (i.e., $K_{\text{from}} = \{a_i\}$), K_{opt} is acquainted with K_{dis} and thus a_i . Hence, the weak connectivity is maintained because K_{opt} keeps its acquaintance with K_{to} (due to a_i 's integration into K_{to}) and becomes acquainted with N .

Otherwise, K_{dis} is the result of extracting a_i from K_{from} . Because of the procedure of removing agents from partitions explained in the previous subsection, we can be sure that a_i and K_{from} are weakly connected: If a_i was an internal node, a_i is acquainted with K_{from} ; if a_i was a leaf, K_{from} is acquainted with a_i . Having integrated a_i , K_{to} is thus weakly connected

with K_{from} . Further, K_{opt} is still acquainted with K_{from} or, due to a_i 's integration, with K_{to} . Finally, the weak connectivity is guaranteed because K_{opt} becomes acquainted with N .

Afterwards, $l(K_{\text{to}})$ proceeds as described in case 1 “ K_{to} is acquainted with a_i ”.

Having integrated a_i into K_{to} , a_i is a leaf with parent a_j in the arborescence representing K_{to} . The application of the candidate move is completed.

Note – Dealing with the Dynamics of MAS

Although we assume that the set of agents \mathfrak{A} participating in solving the PP does not change during SPADA's runtime, SPADA is, in principle, able to deal with agents joining and leaving \mathfrak{A} at runtime. Despite the dynamic nature of MAS, SPADA can ensure a constant number of acquaintances per agent as well as the arborescence property of partitions. However, when agents leave \mathfrak{A} , the weak connectivity of the acquaintances graph cannot be guaranteed without its reinitialization on the basis of global knowledge. We assume that messages are processed correctly and malfunctioning agents or those leaving \mathfrak{A} become unavailable. If a new agent joins \mathfrak{A} , it broadcasts a message and creates unmarked edges to a random subset of agents that reply to this message. Further, some agents replace one of their unmarked edges by a new unmarked edge to the new agent if this does not break the weak connectivity. To identify unavailable agents, the agents periodically ping their acquaintances and their leader. If an agent detects that the head of an unmarked edge is unavailable, it creates a new arbitrary acquaintance. Moreover, if an agent notices that a head of its marked edges is unavailable, it informs its leader which, in turn, removes the unavailable agent from its partition. In case a leader left \mathfrak{A} , each member of the corresponding partition temporarily assumes that it is the leader of a new partition and sends this information to the heads of its marked edges. Agents that receive such a message adopt the sender as leader and forward the message to the heads of their marked edges, thereby recreating a new arborescence in a top-down manner.

8.5 Evaluation and Comparison to PSOPP

In our evaluation, we analyze SPADA's performance with regard to the objectives strict partitioning clustering (C), anticlustering (AC), as well as homogeneous partitioning mean (HPm) and sum (HPs). Since we are particularly interested in SPADA's performance in the context of multi-objective optimization, we also evaluate different combinations of these objectives. Based on different parametrizations of the partitioning constraints, we further examine the influence of dense and sparse solution spaces on SPADA's behavior. By varying the number of agents participating in the partitioning problem, we additionally investigate its ability to find high-quality solutions in search spaces of different size.

Hereinafter, we compare SPADA and PSOPP (see Chapter 7) because they can solve *exactly* the same problems. In particular, both algorithms guarantee that solutions comply with the specified partitioning constraints. In SPADA's evaluation, we therefore deliberately abstain from further comparisons to related approaches that proved to be unsuited in Sections 6.5 and 7.6. PSOPP serves as an adequate benchmark because it is a centralized optimizer whose particles can make use of global knowledge. Moreover, PSOPP benefits from parallelization as the particles concurrently explore the search space. Before we present our evaluation results, we introduce the test bed.

Test Bed

Our implementation of SPADA resides in a simulation environment that uses a sequential, round-based execution model. With respect to Figure 8.1, each optimization cycle corresponds to a specific round. In each optimization cycle, the agents' optimization steps are performed in an arbitrary sequential order that might change from one optimization cycle to another. Consequently, SPADA does not profit from parallelization in its current implementation. Our evaluation results show that this potential drawback does not prevent SPADA from obtaining better results than PSOPP in the context of multi-objective optimization, though.

Problem Structure		n	Size of Partitions		No. of Partitions		Dimensions
			s_{min}	s_{max}	n_{min}	n_{max}	
	PS-01	500	1	n	1	n	1
	PS-02	1000	1	n	1	n	1
	PS-03	2000	1	n	1	n	1
	PS-04	1000	2	20	1	n	1
	PS-05	1000	7	20	1	n	1
	PS-06	1000	14	20	1	n	1
	PS-07	1000	2	20	123	427	1
	PS-08	1000	2	20	196	354	1
	PS-09	1000	2	20	270	280	1
	PS-10	1000	20	50	20	40	6
PS-11	4000	40	100	40	80	6	

Table 8.1: The 11 different problem structures considered for SPADA’s evaluation. Each problem structure defines the number n of agents participating in the partitioning problem, the parametrization of the partitioning constraints, as well as the dimensions of the optimization problem (i.e., the number of objectives).

We analyze SPADA’s performance with regard to different numbers n of agents \mathfrak{A} participating in the partitioning problem as well as different parametrizations of the partitioning constraints. In detail, we evaluated 11 different problem structures. These are listed in Table 8.1. For each problem structure, we generated 20 initial (flat) system structures that differed in the initial number of partitions (such an initial system structure corresponds to a partitioning – possibly residing in a hierarchy – that is to be reorganized). To minimize the influence of the initial number of partitions on our conclusions, we generated these structures in a way that the initial number of partitions was equidistantly distributed between a lower and an upper bound. If the number of partitions was restricted, we used the minimum n_{min} and the maximum n_{max} number of partitions for these bounds. In case of unrestricted instances of the partitioning problem, we set the lower bound to 2 and the upper bound to $\frac{n}{2}$.

For each generated initial system structure, the set of agents \mathfrak{G} was created in exactly the same randomized manner as in the evaluation of PSOPP. Each agent was equipped with a number of properties that corresponds to the number of objectives combined in the objective function. For each dimension, each agent holds a property from the set $\{0, 1, 2, \dots, n - 1\}$ which is unique in this specific dimension. Again, this can be thought of as a very heterogeneous set of power plants that requires well-considered decisions in order to obtain an adequate partitioning (i.e., AVPPs).

For each evaluation scenario (that is, a combination of a set of objectives and a problem structure), we performed 100 simulation runs (5 runs for each of the 20 different initial system structures per problem structure). In each run, SPADA’s and PSOPP’s task was to optimize the initial system structure according to the given objectives. In all evaluation scenarios, we used a predefined time limit as termination criterion.

As is the case with the evaluation of PSOPP, we normalize the objectives’ raw values to the interval $[0, 1]$ for better comparability of the obtained solutions (recall that the raw values correspond to the (negative) sum of the squared Euclidean distances in case of the objectives C and AC, and standard deviations in case of the objectives HPm and HPs). For this purpose, we reuse the fitness function $\mathcal{F}_o(v) = 1.0 - \frac{b_o - v}{b_o - w_o}$, which yields a normalized fitness value for a specific raw value v . It is based on calculated worst w_o and best b_o values of v for the objective $o \in \{C, AC, HPm, HPs\}$. For all objectives, SPADA’s and PSOPP’s goal was to maximize the fitness. The higher the fitness, the better the created partitioning.

We use worst and best raw values determined in the context of unrestricted partitionings in all following experiments. In case of restricted partitionings, these serve as lower and upper bounds for the actually obtainable values. Using the same worst and best raw values in all evaluation scenarios improves comparability of the results. The worst raw value w_o of an objective $o \in \{C, AC, HPm, HPs\}$

n	objective C			objective AC		objective HPm			objective HPs	
	500	1000	4000	1000	4000	1000	2000	4000	1000	4000
w_o	1.04E7	8.33E7	5.33E9	0.00	0.00	499.00	999.00	1999.00	3.53E5	5.66E6
b_o	0.00	0.00	0.00	8.33E7	5.33E9	0.00	0.00	0.00	0.00	0.00

Table 8.2: Overview of the worst and the best raw values used to calculate normalized fitness values for the different objectives $o \in \{C, AC, HPm, HPs\}$ and for different numbers n of agents participating in the partitioning problem.

is calculated as described in Section 7.6. With regard to the best raw value b_o , we have 0 for the objectives C, HPm, and HPs because the agents can form the grand coalition in case of HPm and HPs (which yields a standard deviation of 0), and partitionings in which all partitions are singletons in case of C (yielding a sum of squared Euclidean distances of 0). While the grand coalition is not the only optimal solution for HPm and HPs in this setting, a partitioning algorithm has to create singletons in order to obtain an optimal result for C. Recall that we have $w_{AC} = b_C$ and $b_{AC} = w_C$ for objective AC. Table 8.2 gives an overview of the worst and the best raw values used to derive the normalized fitness values $\mathcal{F}_o(v)$.

With regard to the algorithms' parametrization, we used the parameters we found in Section 7.6 for PSOPP. Suitable parametrizations of SPADA were identified in the course of extensive preceding experiments. Where not stated otherwise, we used the following parameters for SPADA:

- For the number of acquaintances per agent, we used $\#ACQ = 10$.
- The number of transaction partners $\#TP$ of a leader (i.e., the size of the regional partitioning to optimize) is defined as a multiple of the size of its partition. Here, we used a factor of 0.05 for objective C and 2.60 for HPm.
- For the creation of candidate transactions, leaders regarded the 20 % (15 %) best candidate moves for integration or exclusion in case of C (HPm).
- For the maximum number of iterations $\#ITERS_{\max}$ for the creation of candidate transactions, we used 4 times the number of the best candidate moves for integration and exclusion.
- As for maximum number created candidate transactions $\#CT$, we used 10 for C and 20 for HPm.

For further comparison, we performed all experiments with a procedure that corresponds to a random walk through the solution space (referred to as RDM in the following). To this end, we used another instance of PSOPP that was parametrized with $c_{rdm} = 1.0$, $c_{B_i} = 0.0$, $c_B = 0.0$, and a single particle.

Unrestricted Partitionings

In case of unrestricted partitionings, we have $s_{\min} = 1$ and $s_{\max} = n$ for the minimum and the maximum partition size, and $n_{\min} = 1$ and $n_{\max} = n$ for the minimum and the maximum number of partitions. To compare SPADA's and PSOPP's performance with regard to a single objective, we opted for the two objectives C and HPm. We made this decision on the basis of our findings in Section 7.6. There, C and HPm turned out to be the most challenging objectives when optimizing for a single objective. At the end of this section, we consider the two other objectives AC and HPs in the context of multi-objective optimization.

We evaluated objective C in combination with the problem structures PS-01 and PS-02 from Table 8.1. We observe that PSOPP features a much higher rate of convergence than SPADA (see Figures 8.8a and 8.8b). On the one hand, this can be attributed to the fact that PSOPP is a centralized optimizer that relies on global instead of regional knowledge. On the other hand, PSOPP is a population-based approach and thus has the advantage of maintaining multiple candidate solutions. In our evaluation, PSOPP makes use of 4 particles, meaning that it has information of up to 9 different candidate solutions (4 candidate solutions representing the particles' current positions, 4 best found solutions in the particles' neighborhoods, and, finally, the global best found solution). These factors allow PSOPP to considerably improve the quality of the global best found solution right at the beginning of the optimization. The

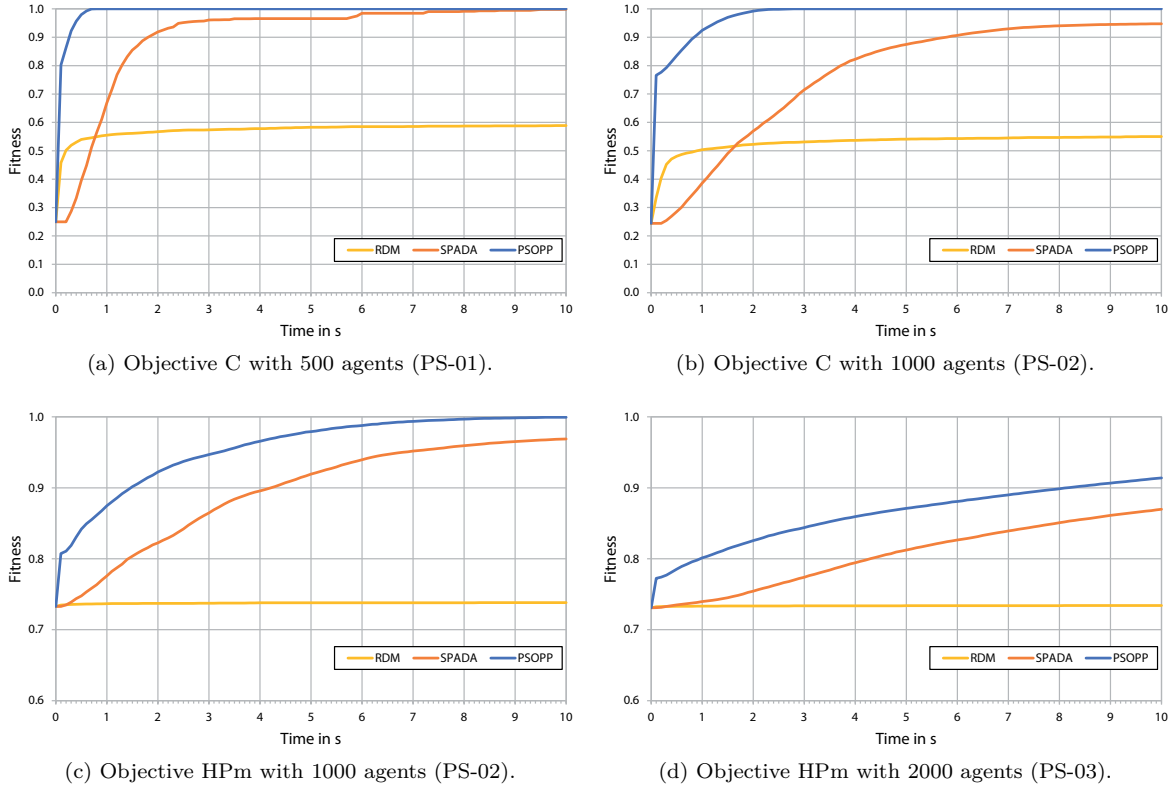


Figure 8.8: Unrestricted partitionings: Development of the fitness values obtained by SPADA, PSOPP, and RDM for objectives C and HPm over a time frame of 10 s. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.

availability of multiple candidate solutions is especially beneficial if there is a high risk of getting trapped in local optima, as is the case with objective C. In contrast to PSOPP, SPADA holds only a single candidate solution that is modified by the agents in the hope of finding an even better solution. In Figure 8.8a, the small jumps in SPADA’s fitness after approximately 2.5 s, 6 s, and 7 s point to situations in which SPADA was able to leave local optima. For the same reasons, SPADA’s rate of convergence decreases more than PSOPP’s when increasing the number of agents from 500 to 1000. While SPADA is able to yield almost optimal partitionings after 8 s for 500 agents and after 10 s for 1000 agents, PSOPP achieves comparable values after much shorter runtimes of 0.6 s and 1.2 s, respectively. After 10 s, PSOPP yields optimal partitionings consisting of 500 (1000) partitions for 500 (1000) agents in all runs. On average, SPADA creates partitionings with 475.42 partitions (standard deviation $\sigma = 107.68$) for 500 agents and 835.78 partitions ($\sigma = 253.83$) for 1000 agents. While SPADA detects that forming singletons is advantageous in this setting, some agents are still grouped with others. This is why it did not always come up with optimal results. Both algorithms achieve significantly better solutions than RDM. Because RDM features a better rate of convergence than SPADA during the first 0.3 s, it might be advantageous to allow the agents to perform a random walk for a short time frame after SPADA’s initialization.

As for objective HPm, we used the problem structures PS-02 and PS-03 (see Table 8.1). We note that SPADA’s and PSOPP’s rates of convergence are much more similar than in case of objective C (see Figures 8.8c and 8.8d). Again, PSOPP exhibits a strong increase in fitness right at the beginning of the optimization and is able to keep this lead until termination. After 10 s, PSOPP created an

Evaluation Scenario	objective C 500 agents (PS-01)	objective C 1000 agents (PS-02)	objective HPm 1000 agents (PS-02)	objective HPm 2000 agents (PS-03)
SPADA	0.999 (0.005)	0.947 (0.219)	0.968 (0.025)	0.858 (0.097)
PSOPP	1.000 (0.000)	1.000 (0.000)	1.000 (0.003)	0.914 (0.100)

Table 8.3: Unrestricted partitionings: Mean fitness values obtained by SPADA and PSOPP for different combinations of objectives and problem structures after a runtime of 10 s. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.

average number of 2.16 partitions ($\sigma = 7.67$) for 1000 agents and 236.83 partitions ($\sigma = 264.36$) for 2000 agents. SPADA established, on average, 219.13 partitions ($\sigma = 112.85$) for 1000 agents and 494.87 partitions ($\sigma = 290.87$) for 2000 agents. As mentioned before, creating the grand coalition is not the only optimal partitioning in this setting. For this reason, both algorithms are able to create high-quality partitionings with a very different number of partitions. However, PSOPP can provide solutions whose quality is comparable to those SPADA obtains at the end of the optimization process already after 4.0 s for 1000 agents and after 4.8 s for 2000 agents. Again, both algorithms significantly outperform RDM.

Table 8.3 lists the fitness values SPADA and PSOPP achieve for the different evaluation scenarios after 10 s. Although SPADA uses only a single candidate solution and regional knowledge to solve the PP, the quality of its solutions for C and HPm is 94 % of the one provided by PSOPP.

Restricted Size of Partitions

As explained in Section 8.3, leaders only require local knowledge to ensure that a partitioning complies with the minimum- and maximum-partition-size constraints. In the following, we examine in which way such restrictions influence SPADA’s performance. We expect that the minimum partition size s_{min} has a stronger influence on SPADA’s performance than the maximum partition size s_{max} because large values for s_{min} hamper the creation of candidate transactions that yield partitions of sufficient size. For this reason, we regard three different restrictions of $s_{min} \in \{2, 7, 14\}$ in combination with $s_{max} = 20$. In detail, we consider the problem structures PS-04, PS-05, and PS-06 listed in Table 8.1. All of them consist of 1000 agents and do not restrict the minimum and the maximum number of partitions (i.e., $n_{min} = 1$ and $n_{max} = 1000$). To investigate the impact of the different parametrizations of s_{min} , we employed objective C because it prefers small partitions in our setting.

In contrast to the evaluation scenarios with unrestricted partitionings, SPADA has to make use of its ability to correct candidate transactions to be able to find high-quality solutions that abide by the partitioning constraints. The success rate of enlarging too small partitions during the local correction of candidate transactions increases with the number of available candidate moves (see Section 8.3). We therefore raised the factor defining the number of a leader’s transaction partners $\#TP$ dependent on the size of its partition from 0.05 to 1.50.

We notice that increasing s_{min} significantly reduces the number of high-quality solutions in the search space. This is indicated by the considerable drop in the mean fitness value of the solutions provided by RDM from 0.548 for $s_{min} = 2$ over 0.177 for $s_{min} = 7$ to 0.108 for $s_{min} = 14$ (see Figure 8.9). Hence, we expect that the more restricted the partition size, the lower SPADA’s and PSOPP’s rates of convergence.

As for $s_{min} = 2$ (see Figure 8.9a), SPADA’s behavior in terms of the fitness development is reminiscent of the one it showed in the unrestricted case (see Figure 8.8b). Interestingly, PSOPP’s rate of convergence decreases to a greater extent. These observations can also be made for $s_{min} = 7$. In these cases, SPADA seems to profit from the automatic increase in exploration that originates from the correction of candidate transactions. The tighter the bounds for the partition size, the higher the exploration rate. As in the unrestricted evaluation scenario, SPADA yields solutions whose quality is about 95 % of those provided by PSOPP after the time limit of 10 s. However, PSOPP now needs 5.7 s and 7.7 s to achieve fitness values comparable to those SPADA attains at the end of the optimization process for $s_{min} = 2$ and $s_{min} = 7$, respectively. Recall that this value was only 1.2 s in the unrestricted case. We attribute this to SPADA’s increase in exploration.

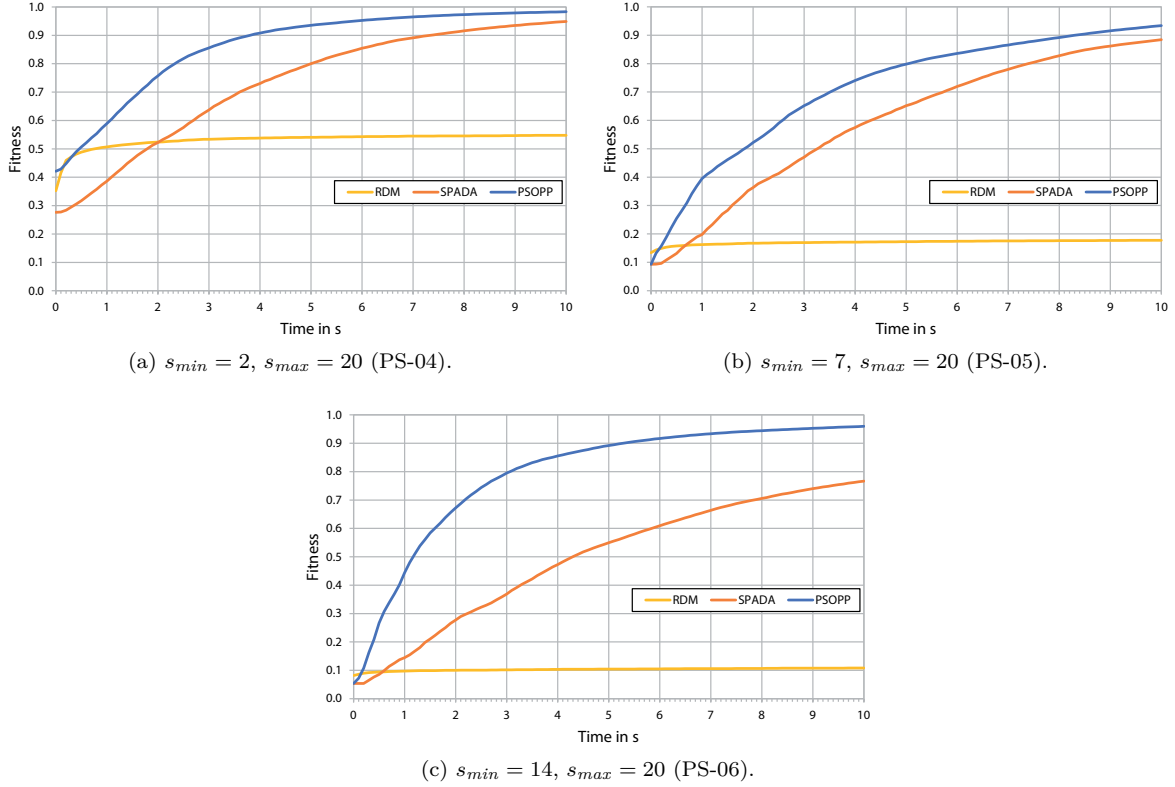


Figure 8.9: Restricted size of partitions: Development of the fitness values obtained by SPADA, PSOPP, and RDM for objective C, 1000 agents, and different restrictions of the minimum and the maximum partition size over a time frame of 10 s. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.

Problem Structure	$s_{min} = 2, s_{max} = 20$ (PS-04)	$s_{min} = 7, s_{max} = 20$ (PS-05)	$s_{min} = 14, s_{max} = 20$ (PS-06)
SPADA	0.932 (0.045)	0.896 (0.060)	0.767 (0.120)
PSOPP	0.983 (0.002)	0.934 (0.012)	0.960 (0.004)

Table 8.4: Restricted size of partitions: Mean fitness values obtained by SPADA and PSOPP for objective C, 1000 agents, and different restrictions of the minimum and the maximum partition size after a runtime of 10 s. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.

Interestingly, PSOPP yields better solutions when using $s_{min} = 14$ instead of $s_{min} = 7$ (SPADA’s solution quality declines with s_{min}). Accordingly, PSOPP’s rate of convergence increases substantially. Here, PSOPP seems to benefit from the reduced size of the search space. We cannot make these observations for SPADA, which we ascribe to SPADA’s difficulty in creating new partitions if s_{min} is large. We therefore suppose that we could resolve this problem by using a parametrization that allows for a higher number of candidate moves and candidate transactions. For $s_{min} = 14$, SPADA’s solution quality is 80 % of PSOPP’s. PSOPP needs 2.7 s to obtain such solutions.

Table 8.4 provides an overview of the fitness values obtained by SPADA and PSOPP after 10 s.

Restricted Size and Number of Partitions

In Section 8.3, we outlined that SPADA ensures compliance with the global number-of-partitions constraints. After a one-time centralized correction, SPADA guarantees the compliance on the basis of local information. In the following, we first investigate whether our principle of localizing the number-of-partitions constraints outperforms a centralized approach to the correction of partitionings. Afterwards, we compare the influence of the number-of-partitions constraints on SPADA’s performance and compare the results to those obtained by PSOPP. As with the restricted size of partitions, we ran experiments for objective C.

Comparison of the Local to the Centralized Approach to Comply with the Specified Number of Partitions For the centralized correction of partitionings, we employed the same procedure SPADA uses for the one-time centralized correction necessary to localize the number-of-partitions constraints. As it would be computationally inefficient to perform the centralized correction after each optimization step, we decided for its application after each optimization cycle. Recall that the centralized correction transforms an invalid partitioning into a valid partitioning by means of minimally invasive changes. In spite of that, it is unsure how these changes affect the partitioning’s fitness. By contrast, our localized principle of complying with the constraints allows the leaders to repair multiple candidate transactions and to choose the best transaction increasing the fitness of their regional partitioning for application. We therefore claim that our local correction not only provides better partitionings but also more stable results in terms of the fitness’s standard deviation. To validate this claim, we ran experiments for problem structure PS-09 (1000 agents, and $s_{min} = 2$, $s_{max} = 20$, $n_{min} = 270$, and $n_{max} = 280$ as parametrization of the partitioning constraints, cf. Table 8.1).

Our results confirm the expected advantages of localizing the global number-of-partitions constraints, which allows for a more target-oriented optimization of partitionings than the centralized approach. In case of our local correction, SPADA is able to steadily increase the fitness until the optimization terminates after 15 s (see Figure 8.10a). That is because the local correction allows the leaders to select those candidate transactions for application that promise the highest increase in the partitioning’s fitness. The one-time centralized correction of the partitioning is performed after a runtime of approximately 5.0 s, which is reflected in the plateau of more or less constant fitness values between 5.0 s and 6.5 s. After 15 s, the local correction yields an average fitness value of 0.986 with a very low standard deviation of $\sigma = 0.005$.

By contrast, the centralized correction is not able to further increase the fitness after a runtime of 9.5 s. The fitness oscillates around a value of 0.867. We ascribe this to the application of candidate transactions that temporarily increase the fitness but actually produce invalid partitionings. The subsequent centralized correction of a partitioning is detrimental to its fitness. As opposed to our local correction that ensures compliance with the number-of-partitions constraints in each optimization step, the centralized correction requires a final correction of the partitioning before it is returned as solution. Although the centralized correction yields a maximum mean fitness of 0.879 (note that the associated partitionings might have been invalid), the actually feasible solution that is returned after the final correction only achieves an average fitness of 0.702 ($\sigma = 0.032$).

With respect to the development of the standard deviation of the fitness depicted in Figure 8.10b, we observe that the local correction of candidate transactions yields a far more stable solution quality than the centralized correction. In case of the local correction, the standard deviation increases until the one-time centralized correction. This can be regarded as an exploration phase. Afterwards, the local correction ensures that the partitioning constraints hold in every optimization step, which is why the standard deviation decreases continuously. Using the centralized correction, the standard deviation remains at a relatively high level until SPADA terminates. The final correction of the partitioning present after 15 s significantly reduces the standard deviation of the actual solution that abides by the partitioning constraints, though.

Comparison to PSOPP To analyze in which way a restricted number of partitions affects SPADA’s performance, we utilized the problem structures PS-07 ($n_{min} = 123$, $n_{max} = 427$), PS-08 ($n_{min} = 196$,

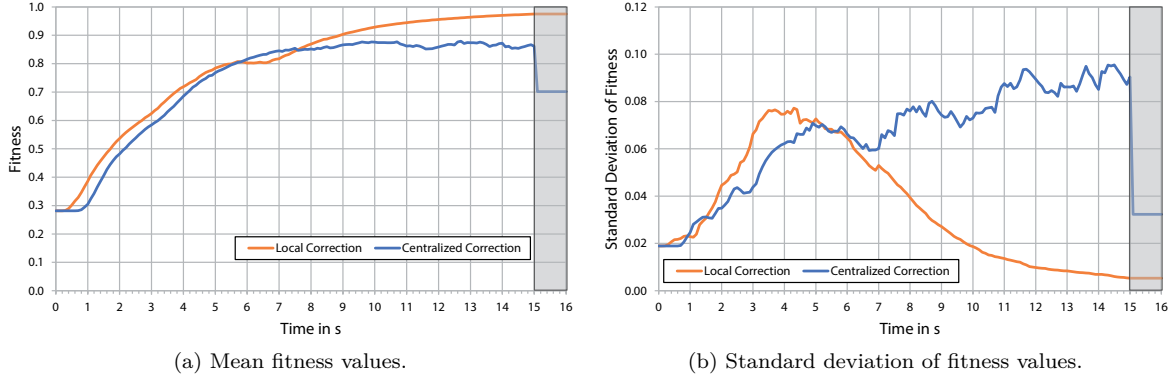


Figure 8.10: Comparison of the local to the centralized correction method: Development of the fitness values (see Figure 8.10a) and the corresponding standard deviations (see Figure 8.10b) obtained by SPADA using either the local correction of candidate transactions or the centralized correction of partitionings. Although we limited SPADA’s runtime to 15 s, we show data for 16 s because the centralized approach has to perform a final correction before returning the solution. The fitness values are averages over 100 runs. In all runs, we used objective C in combination with problem structure PS-09 consisting of 1000 agents ($s_{min} = 2$, $s_{max} = 20$, $n_{min} = 270$, and $n_{max} = 280$).

$n_{max} = 354$), and PS-09 ($n_{min} = 270$, $n_{max} = 280$). All these problem structures consist of 1000 agents and set the minimum and the maximum partition size to $s_{min} = 2$ and $s_{max} = 20$.

As indicated by the development of the mean fitness for RDM, tighter bounds for the number of partitions lead to search spaces that contain less high-quality solutions (see Figure 8.11). After 10 s, RDM achieves mean fitness values of 0.494 for PS-07, 0.422 for PS-08, and 0.345 for PS-09. In case of an unrestricted number of partitions, RDM obtained a fitness of 0.548 for structure PS-04 (see Figure 8.9a).

First of all, we observe that SPADA’s rate of convergence is slightly higher when using a broader range for the number of partitions (see Figure 8.11). That is because the one-time centralized correction of the number of partitions can temporarily impair the quality of a partitioning. The tighter the bounds for the number of partitions, the more changes are necessary, and thus the more likely the one-time centralized correction causes a temporary decline in fitness.

PSOPP’s rate of convergence is similarly resistant to restrictions of the number of partitions as SPADA. After the time limit of 15 s, PSOPP obtains slightly higher fitness values than SPADA (see Table 8.5). The quality of SPADA’s solutions now ranges between 98 % and 99 % of PSOPP’s. PSOPP needs 12.0 s, 12.5 s, and 14.0 s to obtain SPADA’s final fitness values for PS-07, PS-08, and PS-09, respectively. While we observe that PSOPP is less sensitive to restrictions of the number of partitions than SPADA, SPADA seems to benefit from tighter bounds. SPADA can improve the fitness from 0.932 ($\sigma = 0.045$) to 0.986 ($\sigma = 0.005$), compared to the setting where the number of partitions was not restricted (problem structure PS-04). We attribute this to the higher degree of exploration originating from the correction of candidate transactions when using tighter bounds as well as the accompanying reduction of the size of the search space. Note that objective C favors small partitions and all regarded problem structures allow for a relatively high number of partitions.

Multi-Objective Optimization with Restricted Partitionings

Last but not least, we investigate SPADA’s behavior in case of partitioning problems that involve restricted partitionings as well as multiple objectives. This corresponds to a setting as imposed by our power management case study, where the task is to partition a set of power plants into a set of AVPPs on the basis of several formation criteria (see Section 6.3). In this context, the partitioning constraints are used to guide the self-organized hierarchical decomposition of the scheduling problem (see Section 6.2).

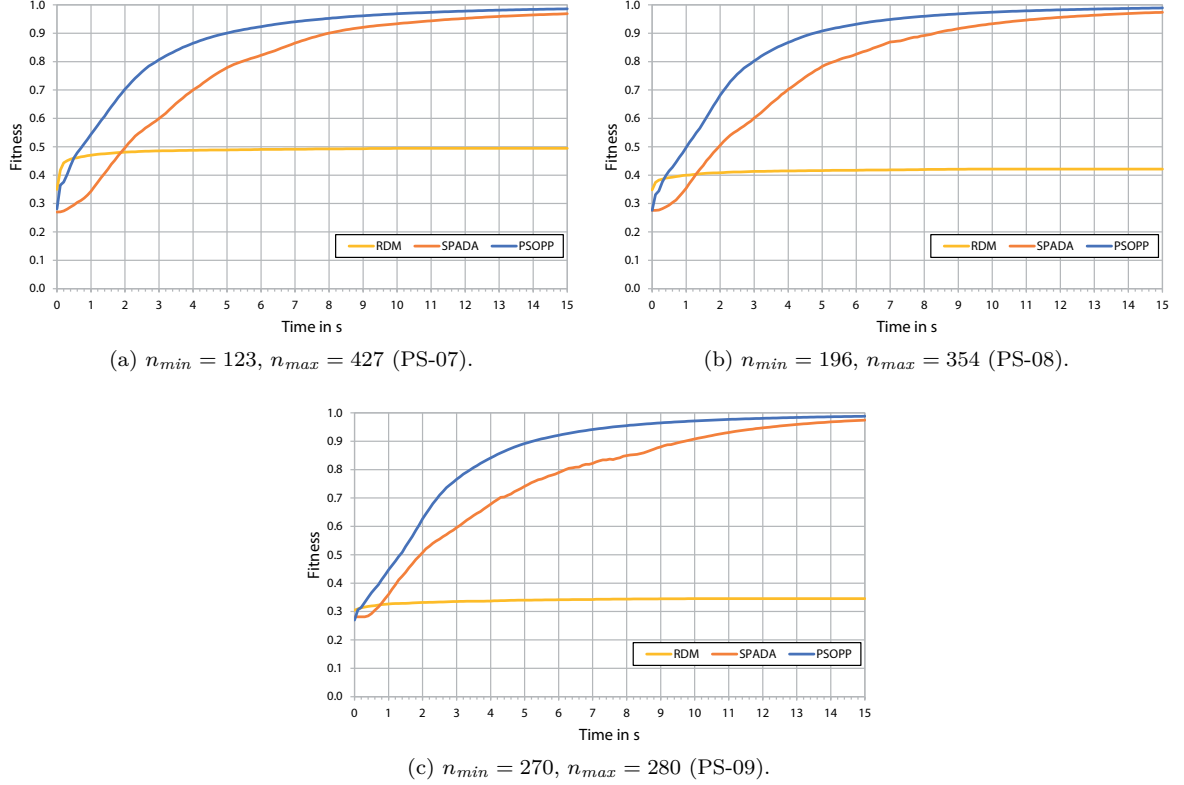


Figure 8.11: Restricted size and number of partitions: Development of the fitness values obtained by SPADA, PSOPP, and RDM for objective C, 1000 agents, and different restrictions of the number-of-partitions constraints over a time frame of 15 s. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.

Problem Structure	$n_{min} = 123,$ $n_{max} = 427$ (PS-07)	$n_{min} = 196,$ $n_{max} = 354$ (PS-08)	$n_{min} = 270,$ $n_{max} = 280$ (PS-09)
SPADA	0.978 (0.014)	0.983 (0.006)	0.986 (0.005)
PSOPP	0.992 (0.001)	0.994 (0.001)	0.994 (0.001)

Table 8.5: Restricted size and number of partitions: Mean fitness values obtained by SPADA and PSOPP for objective C, 1000 agents, and different restrictions of the number-of-partitions constraints after a runtime of 15 s. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.

To assess the fitness of a candidate solution, we use an a priori prioritization of the different objectives, just as we did in the evaluation of PSOPP (see Section 7.6). In detail, a candidate solution's fitness is the average of the fitness values for the single dimensions (each dimension was thus equally weighted). In a preceding parameter search, we identified the following parametrization for SPADA (we only list those values that differ from the parametrization listed before):

- We set the factor defining the number of transaction partners $\#TP$ dependent on a leader's partition size to 10.
- For the creation of candidate transactions, leaders regarded the 30 % best candidate moves for integration or exclusion.
- We allowed leaders to create a maximum of $\#CT = 20$ candidate transactions.

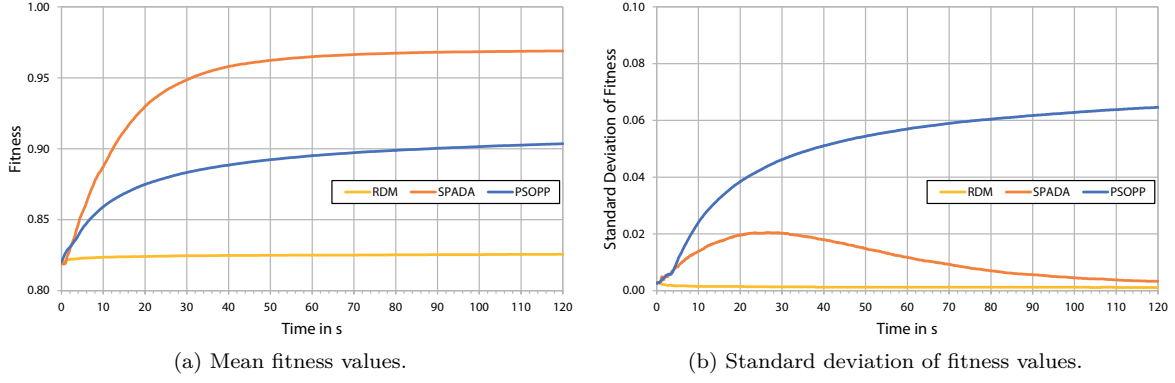


Figure 8.12: First multi-objective evaluation scenario: Development of the fitness values (see Figure 8.12a) and the corresponding standard deviations (see Figure 8.12b) attained by SPADA, PSOPP, and RDM over a time frame of 120s. Results are obtained for the problem structure PS-10 (1000 agents, $s_{min} = 20$, $s_{max} = 50$, $n_{min} = 20$, and $n_{max} = 40$) in a multi-objective setting combining one objective for C, one objective for HPm, and four objectives for HPs. The fitness values are averages over 100 runs. In case of PSOPP, the data is based on the global best found solution.

Note that these parameters enlarge the size of the leaders' regional search space. That way, we increase the chances that leaders can create and apply candidate transactions that improve the fitness of multiple objectives at the same time.

In our following experiments, we utilize the two problem structures PS-10 and PS-11 (see Table 8.1). PS-10 consists of 1000 agents and uses a minimum and maximum partition size of $s_{min} = 20$ and $s_{max} = 50$. The range for the allowed number of partitions is defined by $n_{min} = 20$ and $n_{max} = 40$. We suppose that this parametrization of the partitioning constraints leads to an adequate hierarchical structure. In terms of our case study, PS-10 restricts the maximum size of AVPP to $s_{max} = 50$ power plants. When creating a new layer of AVPPs, a solution to the corresponding partitioning problem contains at most $n_{max} = 40$ new AVPPs that have to be integrated below an existing AVPP in the hierarchy. PS-11 considers the case of 4000 agents, which is why we increase the boundaries for the size and the number of partitions to $s_{min} = 40$, $s_{max} = 100$, $n_{min} = 40$, and $n_{max} = 80$. Both problem structures regard a combination of 6 objectives. Based on these problem structures, we created three different evaluation scenarios. We suppose that the presence of multiple objectives leads to a decline in SPADA's as well as PSOPP's rate of convergence.

First Multi-Objective Evaluation Scenario In our first multi-objective evaluation scenario, we combined PS-10 with one objective for clustering (C), one objective for homogeneous partitioning mean (HPm), and four objectives for homogeneous partitioning sum (HPs). For PSOPP, we used the parameters $c_{rdm} = 0.2$, $c_{B_i} = 0.0$, and $c_B = 0.8$ that we identified for the combination C-HPm-HPs in Section 7.6. Due to our findings in PSOPP's evaluation, we hypothesize that the objectives C and HPm have the strongest influence on the composition of the created partitioning.

As expected, the rate of convergence decreases for all considered algorithms (see Figure 8.12a). The development of the fitness obtained by RDM, which remains more or less at the same level over the entire time frame of 120s, highlights the need for a systematic search for high-quality solutions. In contrast to the evaluation scenarios with a single objective, SPADA now achieves substantially higher fitness values than PSOPP for time limits greater than 2.5s. SPADA needs only 13s to achieve the fitness PSOPP reaches after a runtime of 120s. On average, PSOPP yields solutions whose quality is about 93 % of those provided by SPADA. In detail, SPADA obtains an average fitness of 0.969 ($\sigma = 0.003$), compared to 0.904 ($\sigma = 0.065$) in case of PSOPP. As depicted in Figure 8.12b, SPADA not only calculates better solutions on average, but also more reliably since the standard deviation of PSOPP's solutions is 20 times

Dimension	Dim. 1: obj. C	Dim. 2: obj. HPm	Dim. 3: obj. HPs	Dim. 4: obj. HPs	Dim. 5: obj. HPs	Dim. 6: obj. HPs	Total Fitness
SPADA	0.830 (0.016)	0.998 (0.002)	0.996 (0.002)	0.996 (0.001)	0.997 (0.001)	0.996 (0.001)	0.969 (0.003)
PSOPP	0.463 (0.369)	0.979 (0.015)	0.995 (0.002)	0.995 (0.002)	0.995 (0.002)	0.995 (0.002)	0.904 (0.065)

Table 8.6: First multi-objective evaluation scenario: Overview of the mean fitness values attained by SPADA and PSOPP for the six dimensions of a multi-objective setting after a runtime of 120 s. The set of objectives comprises one objective for C, one objective for HPm, and four objectives for HPs. It is performed on the basis of problem structure PS-10. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.

higher after the time limit of 120 s.

When taking a look at the fitness values for the single dimensions (see Table 8.6), we notice that SPADA achieves much better solutions for objective C (0.830 with $\sigma = 0.016$ compared to 0.463 with $\sigma = 0.369$) and slightly better solutions for objective HPm (0.998 with $\sigma = 0.002$ compared to 0.979 with $\sigma = 0.015$). We attribute this to the regionalized optimization principle in which the agents iteratively refine the composition of overlapping regional partitionings from different perspectives. Together with SPADA’s ability to benefit from synergy effects, this principle reduces the risk of getting trapped in local optima. Furthermore, we presume that PSOPP is more afflicted by sparse solution spaces in a multi-objective context. With regard to the four HPs objectives, SPADA and PSOPP obtain a similar quality. As shown in Section 7.6, it is relatively easy to find good solutions for HPs, which is why RDM also provides solutions with relatively high fitness values (see Figure 8.12a).

Second Multi-Objective Evaluation Scenario In our second multi-objective evaluation scenario, we reused PS-10 but, compared with the previous evaluation scenario, we replaced objective C with an objective for anticlustering (AC). For PSOPP, we used the parameters $c_{rdm} = 0.2$, $c_{\mathcal{B}_i} = 0.1$, and $c_{\mathcal{B}} = 0.7$ that we identified for the combination AC-HPm-HPs in Section 7.6. In this evaluation scenario, we expect that SPADA reaches slightly higher fitness values than in the first multi-objective evaluation scenario. We further suppose that PSOPP creates partitionings whose quality is closer to those obtained by SPADA than before because AC is less prone to local optima than C.

Figure 8.13 shows the expected behavior. PSOPP is able to obtain a high fitness value of 0.990 after 9.4 s. Over the remaining 53.8 s, PSOPP is not able to substantially improve its solutions, though. This is why its final solutions still have an average fitness of 0.990 ($\sigma = 0.004$) after reaching the time limit of 60 s. SPADA provides the same average quality already after 5.5 s and is even able to slightly increase the fitness to an average of 0.998 ($\sigma = 0.004$). In this situation, PSOPP yields solutions whose quality is about 99 % of those provided by SPADA.

These findings are also reflected by the fitness values for the single dimensions provided in Table 8.7: SPADA and PSOPP achieve similar fitness values for objective AC (0.998 with $\sigma = 0.001$ compared to 0.993 with $\sigma = 0.004$) and, again, more or less identical values for the four HPs dimensions. The differences concerning objective HPm are most noticeable but still comparable to those observed in the first multi-objective evaluation scenario. These observations confirm that SPADA is well suited for solving the partitioning problem in the context of constrained partitionings and a combination of multiple objectives.

Third Multi-Objective Evaluation Scenario In our third and last multi-objective evaluation scenario, we used the same combination of objectives as in the first multi-objective evaluation scenario, but, this time, in combination with problem structure PS-11 that consists of 4000 agents and calls for larger and more partitions ($s_{min} = 40$, $s_{max} = 100$, $n_{min} = 40$, and $n_{max} = 80$). Because the increase in the number of agents leads to a significantly larger search space, we hypothesize that, in comparison to the first multi-objective evaluation scenario, the differences in the fitness values obtained by SPADA and PSOPP become more prominent.

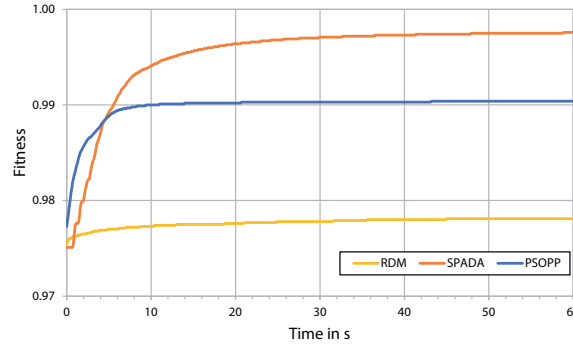


Figure 8.13: Second multi-objective evaluation scenario: Development of the fitness values attained by SPADA, PSOPP, and RDM over a time frame of 60 s. Results are obtained for the problem structure PS-10 (1000 agents, $s_{min} = 20$, $s_{max} = 50$, $n_{min} = 20$, and $n_{max} = 40$) in a multi-objective setting combining one objective for AC, one objective for HPm, and four objectives for HPs. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.

Dimension	Dim. 1: obj. AC	Dim. 2: obj. HPm	Dim. 3: obj. HPs	Dim. 4: obj. HPs	Dim. 5: obj. HPs	Dim. 6: obj. HPs	Total Fitness
SPADA	0.998 (0.001)	0.999 (0.000)	0.997 (0.001)	0.997 (0.001)	0.997 (0.001)	0.997 (0.001)	0.998 (0.000)
PSOPP	0.993 (0.004)	0.975 (0.015)	0.994 (0.002)	0.994 (0.002)	0.994 (0.002)	0.994 (0.002)	0.990 (0.004)

Table 8.7: Second multi-objective evaluation scenario: Overview of the mean fitness values attained by SPADA and PSOPP for the six dimensions of a multi-objective setting after a runtime of 60 s. The set of objectives comprises one objective for AC, one objective for HPm, and four objectives for HPs. It is performed on the basis of problem structure PS-10. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.

Figure 8.14 confirms our expectations. The increased size of the search space causes a decline in the rate of convergence of both SPADA and PSOPP. After a runtime of 120 s, SPADA and PSOPP obtain a mean fitness of 0.886 and 0.840, respectively. Note that these values are significantly lower than those obtained in the first multi-objective evaluation scenario where SPADA and PSOPP obtained a mean fitness of 0.969 and 0.904, respectively. SPADA is able to improve the mean fitness to 0.952 ($\sigma = 0.026$) using the full runtime of 400 s, though. The increase in fitness provided by PSOPP turns out to be lower (mean fitness of 0.856 with $\sigma = 0.029$). Apart from objective C, the fitness values for the single dimensions are almost identical to those achieved in the first multi-objective evaluation scenario where the search space was defined by 1000 instead of 4000 agents (see Table 8.8). The fact that SPADA can keep the fitness for objective C at a moderate value of 0.733 ($\sigma = 0.155$) – compared to 0.167 ($\sigma = 0.167$) in case of PSOPP – again demonstrates the advantage of SPADA’s regionalized optimization principle in a multi-objective setting. On average, PSOPP creates solutions whose quality is about 90 % of those provided by SPADA.

Summary of the Evaluation Results

To summarize, PSOPP outperforms SPADA in case of single objectives where its centralized and population-based approach helps in exploiting specific characteristics of the fitness landscape, such as gradients. On the other hand, SPADA finds superior partitionings in the presence of multiple objectives, even in sparse solution spaces. In such situations and especially in large search spaces, SPADA benefits from its regionalized optimization principle that reduces the risk of getting trapped in local optima.

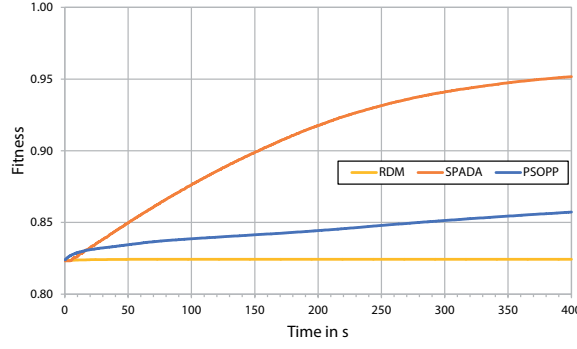


Figure 8.14: Third multi-objective evaluation scenario: Development of the fitness values attained by SPADA, PSOPP, and RDM over a time frame of 400 s. Results are obtained for the problem structure PS-11 (4000 agents, $s_{min} = 40$, $s_{max} = 100$, $n_{min} = 40$, and $n_{max} = 80$) in a multi-objective setting combining one objective for AC, one objective for HPm, and four objectives for HPs. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.

Dimension	Dim. 1: obj. C	Dim. 2: obj. HPm	Dim. 3: obj. HPs	Dim. 4: obj. HPs	Dim. 5: obj. HPs	Dim. 6: obj. HPs	Total Fitness
SPADA	0.733 (0.155)	0.997 (0.002)	0.996 (0.001)	0.996 (0.001)	0.996 (0.001)	0.996 (0.001)	0.952 (0.026)
PSOPP	0.167 (0.167)	0.978 (0.013)	0.997 (0.001)	0.997 (0.001)	0.997 (0.001)	0.997 (0.001)	0.856 (0.029)

Table 8.8: Third multi-objective evaluation scenario: Overview of the mean fitness values attained by SPADA and PSOPP for the six dimensions of a multi-objective setting after a runtime of 400 s. The set of objectives comprises one objective for C, one objective for HPm, and four objectives for HPs. It is performed on the basis of problem structure PS-11. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.

Moreover, our evaluation highlighted that SPADA’s approach to localize the global number-of-partitions constraints should be preferred to a centralized correction of partitionings.

Chapter Summary and Outlook

In this chapter, we presented a decentralized agent-based algorithm, called SPADA, that solves the partitioning problem (PP) introduced in Section 6.1 in an iterative and general manner. In contrast to our centralized particle swarm optimizer PSOPP (see Chapter 7), SPADA enables the group of agents that is to be partitioned to solve the PP by themselves. To cope with large search spaces, the agents make use of an overlay network to autonomously decompose the overall PP into overlapping sub-problems. Each of these sub-problems corresponds to the task of optimizing the composition of a specific regional partitioning, i.e., to improve a partial solution to the overall PP. This is achieved in a regio-centralized way. Because the regarded sub-problems change from one iteration to another, the optimization is carried out from different perspectives so that a high-quality overall solution evolves over time. This procedure reduces the attraction of local optima, which is of particular interest in multi-objective optimization problems and constrained search spaces. The latter results from the partitioning constraints whose satisfaction is guaranteed by SPADA. SPADA is therefore ideally suited for the self-organized creation of hierarchical system structures as explained in Chapter 9. To be able to solve the PP – despite the global number-of-partitions constraints – in a decentralized manner, SPADA decomposes the global constraints into local counterparts whose satisfaction implies adherence to the global constraints. Our evaluation demonstrates that SPADA’s average solution quality ranges between 80 % and 99 % of PSOPP’s in case

of single objectives. In case of multi-objective optimization problems, SPADA outperforms PSOPP. Here its solutions are, on average, up to 11 % better than PSOPP's.

Future work includes the investigation of SPADA's performance in a simulation environment that allows for the concurrent optimization of regional partitionings. Furthermore, we want to examine different possibilities for guiding the optimization of regional partitionings on the basis of the partitioning constraints. This would improve the systematic search for high-quality solutions, e.g., by promoting SPADA's ability to create new partitions despite high values for the minimum size of partitions. In this context, an interesting idea might be to integrate PSOPP's move operations with the creation of candidate transactions for the optimization of regional partitionings. Moreover, we will analyze SPADA's performance in highly dynamic systems in which the set of agents to be partitioned might change at runtime.

Self-Organized Formation of Hierarchical System Structures

Summary. If a group of agents has to solve an optimization problem whose complexity is subject to the number of participating agents and if this problem can be decomposed in a hierarchical manner, hierarchical system structures enable scalability with regard to the number of agents involved. An example of such an optimization problem is the scheduling problem considered in this thesis. A self-organizing hierarchy that can adapt its structure to the current environmental conditions and the system's internal state, allows the agents to come to a compromise between solution quality (in terms of the satisfaction of the demand and costs) and runtime performance themselves. In this chapter, we present a control loop for creating and maintaining adequate hierarchical system structures at runtime. To form stable hierarchies whose structure support the system's goals, the control loop recursively solves partitioning problems by means of a partitioning algorithm, such as PSOPP or SPADA. That way, the system structure is optimized according to application-specific formation criteria. In our evaluation, we show that this scheme establishes hierarchies that, compared to a centralized approach, considerably reduce the time needed to create power plant schedules. We further demonstrate that the concepts of partitioning constraints and homogeneous partitioning devised in this thesis are fundamental ingredients to obtain stable and scalable self-organizing hierarchies that allow for high-quality solutions to the scheduling problem.

Publication. The Hierarchical Control Loop has been introduced in Steghöfer et al. [207]. The evaluation is based on the findings that have been published in Anders et al. [16].

As described in Chapter 6, hierarchical system structures enable scalability with regard to the number of agents participating in the scheduling problem. Our evaluations confirm that deep hierarchies – in which, on average, intermediaries control a lower number of agents – can remarkably decrease the time needed to create schedules. We noticed that short maximum sequential scheduling times come at the expense of solution quality, though. As explained in Section 6.2, this can be ascribed to the observation that deep hierarchies suffer from fragmentation leading to inaccuracies in the intermediaries' abstracted control models. Moreover, increasing the number of subsystems or the hierarchy's height does not necessarily cause shorter max. seq. scheduling times (see Section 9.2). Hence, there is a trade-off between the depth and the width of a hierarchy influencing both runtime performance and solution quality.

Due to the dynamic environment and changes in the agents' behavior, there is a need to establish and maintain suitable hierarchical system structures in response to the current environmental conditions and the system's internal state at runtime. If the hierarchy is created in a self-organizing manner and in compliance with the current circumstances under which the scheduling problem has to be solved, the system is able to find a suitable trade-off between the time needed to create schedules and the solution quality itself. That way, self-organization serves as a dynamic and autonomous form of problem

decomposition that scales with the number of agents schedules have to be created for.

To obtain a self-organizing hierarchy, we reuse the principle of problem decomposition through partitioning a set of agents into several pairwise disjoint partitions, i.e., subsystems, in accordance with application-specific objectives (see Chapter 6). Recall that if this set of agents corresponds to all agents in the system, we gain a flat hierarchy of height two (see Figure 6.1b). In this structure, each created subsystem is represented by an intermediary which controls its subordinate agents. These intermediaries are, in turn, controlled by the top-level intermediary which constitutes the hierarchy's root. Compared to this structure, the centralized system has a height of 1 and all agents are directly controlled by the top-level intermediary (see Figure 6.1a). The partitioning constraints introduced in Section 6.2 are used to specify an appropriate degree of decomposition by prescribing ranges for the number and the size of the subsystems. Using ranges instead of fixed values provides the system with degrees of freedom needed to optimize the partitionings' composition according to application-specific formation criteria. With regard to a specific subsystem, the scheduling problem can be decomposed once more by solving the partitioning problem recursively – but now with respect to the children of the regarded intermediary (in a hierarchy, such a child might be an intermediary itself). Because each created subsystem is again represented by a new intermediary, the height of the corresponding subtree in the hierarchy increases by one, which is why the its leaves do not have to have the same depth. The fact that we solve a partitioning problem when extending the hierarchy ensures that its overall composition complies with the application-specific formation criteria. To maintain suitable partitionings, each intermediary monitors the composition of the partitioning its children are situated in and triggers a reorganization, i.e., a re-partitioning, if the partitioning violates a composition constraint. Note that, in a hierarchy, such a partitioning consists of the intermediary's children and nephews. While this control action optimizes the partitioning's composition, it does not change the hierarchy's height. Because the decomposition of a subsystem into further subsystems as well as the re-partitioning a specific region of the hierarchy requires solving an instance of the partitioning problem, the algorithms PSOPP and SPADA (see Chapters 7 and 8) take on a central role in such a self-organizing hierarchy. The only control action that comes by without solving a partitioning problem is the one that reduces the degree of decomposition within a specific subsystem. In this case, it is sufficient to dissolve an existing intermediary. The height of the corresponding subtree in the hierarchy decreases by one. The dissolution of an intermediary is advantageous if scheduling times are shorter than necessary. Since the degrees of freedom increase, solution quality is likely to be improved. Note that the top-level intermediary must not be dissolved as this would break the hierarchical structure.

The result of these control actions is a self-organizing hierarchy as depicted in Figure 6.1c. In this structure, each intermediary is an internal node and represents the root of a specific subtree. Because each intermediary thus represents an entire subsystem, the overall system constitutes a system of systems. A reorganization might cause an intermediary to switch from one subsystem to another. In such a situation, not only the intermediary but its entire subtree changes its position within the hierarchy. With regard to the case study, intermediaries and leaves correspond to AVPPs and physical power plants, respectively. An example of a hierarchy of AVPPs and physical power plants is shown in Figure 2.1.

The decisions when to dissolve an existing intermediary, when to introduce new intermediaries, or when to reorganize partitionings within the hierarchy are handled by the so-called *Hierarchical Control Loop* (in [207], we referred to the control loop as “HiSPADA” – however, to avoid any confusion with the SPADA algorithm, we do not use the term here). The Hierarchical Control Loop mainly constitutes a *partitioning control* and runs on each intermediary. To take account of existing organizational structures, such as utilities and grid operators, or physical infrastructure, such as the power grid, the Hierarchical Control Loop allows for self-organization on top of partially predefined system structures. To this end, predefined organizational entities are simply represented by *non-dissolvable* intermediaries. The partitioning control creates a sub-hierarchy for each of these predefined organizational entities at runtime. However, if the predefined structure is too rigid, the partitioning control is not able to tackle the scalability issues it is intended for.

In the remainder of this chapter, we introduce the Hierarchical Control Loop for the self-organized formation of hierarchies in Section 9.1 before we discuss our evaluation results in the context of the decentralized power management case study in Section 9.2.

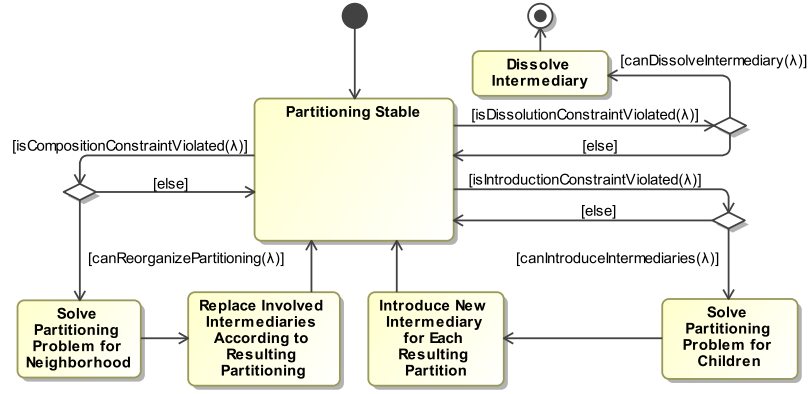


Figure 9.1: The Hierarchical Control Loop as performed by an intermediary λ : The intermediary monitors the violation of specific constraints, depicted as guards on the transitions. In case of a violation, it triggers an associated control action that either introduces new intermediaries, dissolves the intermediary, or reorganizes the partitioning its children belong to. In the latter case, the *neighborhood* denotes the set of agents to be re-partitioned. It consists of the intermediary’s children and nephews. Adapted from [207].

9.1 The Hierarchical Control Loop

The Hierarchical Control Loop’s main purpose is to monitor the intermediary it runs on and to react to the violation of three basic constraints that are depicted as guards on the transitions in Figure 9.1. The *composition constraint* (see Section 6.4) aims at maintaining suitable partitionings within the hierarchy. It is evaluated in the context of the partitioning the intermediary’s children belong to. A violation indicates that the partitioning features an unwanted composition. The satisfaction is re-established by means of a reorganization. The set of affected agents, i.e., those participating in the corresponding partitioning problem, is called the intermediary’s *neighborhood*. It comprises its children and nephews. The other two constraints address the system’s performance in terms of runtime. In case of the violation of the so-called *introduction constraint*, the intermediary triggers the introduction of new intermediaries. Otherwise if the so-called *dissolution constraint* is violated, the intermediary dissolves itself. Both “horizontal” and “vertical” reorganizations are control actions that attempt to restore a violated constraint. This behavior conforms to the Restore Invariant Approach described in Section 1.4. With regard to our case study, the Hierarchical Control Loop runs on each AVPP.

The Hierarchical Control Loop is independent of the concrete partitioning algorithm used. Its only requirements are that it must be possible to (1) limit the partitioning algorithm to an arbitrary set of agents, (2) define application-specific formation as well as termination criteria, and (3) specify partitioning constraints (such as the minimal number of partitions to be formed) to steer the degree of decomposition and thus the hierarchy’s shape. The two partitioning algorithms PSOPP and SPADA meet all of these requirements. In the following, we describe the three control actions triggered by the Hierarchical Control Loop in more detail.

Dissolving Intermediaries

In order to maintain a system structure that features an appropriate balance between scheduling time and solution quality, each intermediary observes the time it needs to create schedules. If the runtime of the scheduling algorithm falls below a given threshold, the corresponding intermediary observes a violation of the *dissolution constraint*. In such a case, the intermediary risks to achieve short runtimes at the cost of solution quality. Before the intermediary is deleted, it transfers its children to its superordinate intermediary (i.e., its parent), which, in turn, assumes control over these agents. Figure 9.2 shows the effect of dissolving an intermediary.

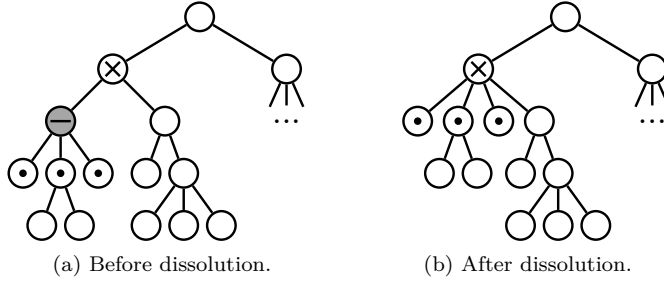


Figure 9.2: Dissolution of an intermediary: The initiating agent is marked in gray. Before its dissolution, it transfers its children “ \odot ” to its parent “ \otimes ”.

Before performing this type of vertical reorganization, the initiating intermediary λ has to check if it can be dissolved under the current circumstances. The predicate `canDissolveIntermediary(λ)` indicates if this is possible:

$$\begin{aligned} \text{canDissolveIntermediary}(\lambda) &:\Leftrightarrow \text{isDissolvable}(\lambda) \\ &\quad \wedge \text{notInReconf}(\text{children}(\lambda) \cup \{\text{parent}(\lambda), \lambda\}) \end{aligned}$$

The predicate checks if λ can be dissolved at all (cf. `isDissolvable(λ)`) and if none of the participating agents is currently involved in another reorganization (cf. `notInReconf(...)`). An intermediary is not dissolvable if it is top-level intermediary Λ (i.e., the hierarchy’s root) or if it represents a predefined organization (e.g., a specific electricity supplier – that is, a non-virtual organization – whose power plants should not be mixed with others). The control action must not be carried out if λ , its parent `parent(λ)`, or its children `children(λ)` participate in another reorganization because interleaved, i.e., interfering, reorganizations might cause an inconsistent organizational structure. The predicate `notInReconf(K)` evaluates to true if and only if none of the agents in K participates in another reorganization:

$$\text{notInReconf}(K) :\Leftrightarrow \forall a \in K : \text{notInReconf}(a)$$

To avoid fluctuations, it is also possible to extend `canDissolveIntermediary(λ)` by a check if a period of grace has expired. This prevents newly created intermediaries from being dissolved right away. The same applies to the predicates `canIntroduceIntermediaries(λ)` and `canReorganizePartitioning(λ)` that become relevant in the two following subsections.

If `canDissolveIntermediary(λ)` holds, the intermediary λ informs its parent and its children about taking part in this vertical reorganization. This measure prevents clashes with other reorganizations by upholding the non-interference property between concurrent reorganizations.

With regard to the scheduling problem, the former children of the dissolved intermediary λ can keep their current schedules until `parent(λ)` creates new ones. This is possible since an intermediary’s scheduled supply is entirely provided by its children. In our power management case study, for instance, an AVPP’s output originates from its underlying physical power plants.

Introducing New Intermediaries

If the time needed to create schedules for subordinate agents exceeds a predefined threshold, an intermediary λ observes a violation of the *introduction constraint*. In such a situation, the intermediary further decomposes the scheduling problem by triggering the partitioning of its children. The corresponding partitioning problem for the set of agents $\mathfrak{A} = \text{children}(\lambda)$ is solved by PSOPP or SPADA with respect to the partitioning constraints and application-specific formation criteria, such as, partitions with a similar number of dispatchable power plants. The minimum number of partitions must be at least two to achieve an actual decomposition (this would not be the case if we permitted the “grand coalition”). Note that it is important to solve a partitioning problem instead of creating partitions at random in order to obtain a stable and efficient problem decomposition and system behavior (cf. Section 6.3). Each resulting partition is represented by a newly created intermediary. These agents become the children of the initiating intermediary and the new parents of its former children. Figure 9.3 illustrates this process.

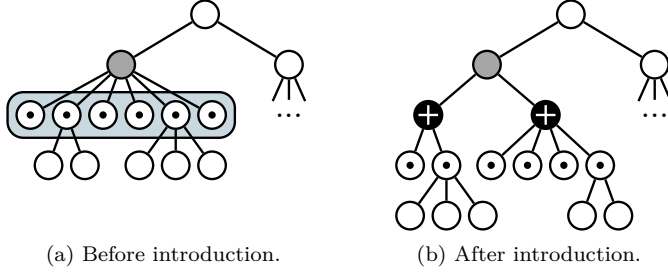


Figure 9.3: Introduction of new intermediaries: The initiating intermediary is marked in gray. Its children – highlighted by the rectangle – participate in a partitioning problem. Each created partition is represented by a new intermediary “ \oplus ” shown in black.

In contrast to the dissolution of an intermediary, this type of vertical reorganization may be triggered by intermediaries that cannot be dissolved:

$$\begin{aligned} \text{canIntroduceIntermediaries}(\lambda) &: \Leftrightarrow \text{canSatisfyPartitioningConstraints}(\text{children}(\lambda)) \\ &\quad \wedge \text{notInReconf}(\text{children}(\lambda) \cup \{\lambda\}) \end{aligned}$$

Note that the control action cannot be performed if the children of intermediary λ do not satisfy Equation (6.3). A violation of Equation (6.3) indicates an incompatible parametrization of the partitioning constraints. In particular, λ has to control a sufficient number of children to be able to form an adequate partitioning that fulfills $|\text{children}(\lambda)| \geq s_{\min} \cdot n_{\min}$. Furthermore, the control action cannot be executed if λ or one of its children are already involved in another reorganization. To preserve the hierarchy’s consistency, λ informs its children about the reorganization if $\text{canIntroduceIntermediaries}(\lambda)$ holds.

Similar to the dissolution of an intermediary, λ ’s former children $\text{children}(\lambda)$ can simply supply resources according to their current schedules until new ones are created. The next time schedules are created, λ assigns schedules to the new intermediaries. To ensure that these schedules comply with their current state and capabilities, each newly created intermediary uses model abstraction to provide λ with an adequate representation of its collective as explained in Chapter 6.

Re-Partitioning an Intermediary’s Neighborhood

To ensure that all partitionings within the hierarchical system structure support the system’s goals, each intermediary monitors if the partitioning its children belong to adheres to the application-specific *composition constraint*. Recall that the composition constraint is a conjunction of the constraints defining the general properties of a partitioning, the partitioning constraints, and constraints subsuming different formation criteria. In the power management case study, one aspect encapsulated in this constraint is the even distribution of uncertainties resulting from non-dispatchable power plants over the AVPPs, which aims at supporting the system’s ability to balance supply and demand (see Section 6.3 for other criteria). The partitioning an intermediary λ observes consists of its children and those nephews that belong to dissolvable siblings. The restriction to dissolvable siblings is necessary to ensure that predefined organizations are not changed – bear in mind that λ and its involved siblings will be replaced as a result of the re-partitioning. We call the set of agents to be re-partitioned λ ’s *neighborhood* $N(\lambda)$:

$$N(\lambda) := \{a \mid a \in \text{children}(\lambda) \vee (\exists \lambda' \in \text{siblings}(\lambda) : \text{isDissolvable}(\lambda') \wedge a \in \text{children}(\lambda'))\}$$

As soon as λ monitors the violation of the composition constraint, it tries to trigger a re-partitioning, that is, a horizontal reorganization, of the agents $\mathfrak{A} = N(\lambda)$. The partitioning to be reorganized is likely to be a reasonable starting point for finding an adequate partitioning fulfilling the formation criteria. For this reason and to promote surgical modifications that re-establish the satisfaction of the composition constraint, it is beneficial to initialize the employed partitioning algorithm with the current partitioning (see Section 6.4). In case of PSOPP, this is achieved by initializing a predefined number of particles with the current partitioning (see Section 7.2). As for SPADA, λ simply initializes the acquaintances graph with the current partitioning (see Section 8.2). Figure 9.4 shows an example of re-partitioning an intermediary’s neighborhood.

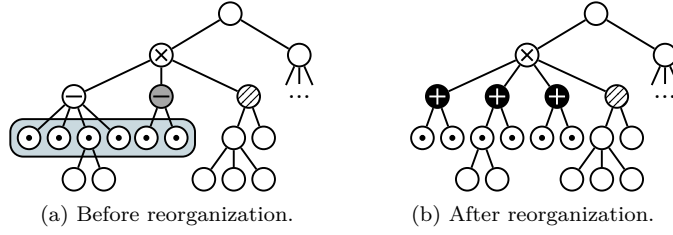


Figure 9.4: Reorganization of an intermediary's neighborhood: The initiating intermediary is marked in gray. Its neighborhood is highlighted by the rectangle. The dashed intermediary represents a predefined organization and cannot be dissolved, which is why its children are excluded from the neighborhood. After the reorganization, the intermediaries “ \ominus ” are replaced by new intermediaries “ \oplus ” shown in black.

After the re-partitioning of λ 's neighborhood $N(\lambda)$, λ and the participating siblings are replaced by new intermediaries representing the new partitioning. Since this not only changes the parent-child relationships of the agents in $N(\lambda)$ but also of λ 's parent, the control action cannot be performed if λ , its parent, one of the agents in $N(\lambda)$, or a participating sibling is already involved in another reorganization. As is the case with the dissolution of an intermediary, the control action further cannot be carried out if λ must not be dissolved:

$$\begin{aligned} \text{canReorganizePartitioning}(\lambda) &:\Leftrightarrow \text{isDissolvable}(\lambda) \\ &\wedge \text{notInReconf}(N(\lambda) \cup \text{siblings}(\lambda) \cup \{\text{parent}(\lambda), \lambda\}) \end{aligned}$$

If $\text{canReorganizePartitioning}(\lambda)$ evaluates to true, λ informs its parent, all participating siblings, and all agents in $N(\lambda)$ about the reorganization to prevent interfering reorganizations.

Similar to the dissolution of an intermediary or the introduction of new ones, all agents in $N(\lambda)$ can stick to their current schedules until the next schedule creation. The next time $\text{parent}(\lambda)$ creates schedules, the newly created intermediaries send the current state as well as the abstraction of the control models of their collectives (cf. Chapter 6).

9.2 Evaluation in the Decentralized Power Management Case Study

Steghöfer [206] provided a first basic evaluation of the Hierarchical Control Loop in the context of our power management case study. In this evaluation, the system was initialized with different predefined hierarchical structures. At a certain point in time, the predictability of the weather-dependent power plants was changed. Due to the initial hierarchical structure and because the disturbance of the predictability was only of temporary nature, the evaluation mainly addressed the following two points: (1) On the one hand, it demonstrated the ability of the Hierarchical Control Loop to trigger a control action in response to the violation of the composition constraint. This violation was caused by the change in the weather-dependent power plants' predictability. The corresponding control action was carried out by the SPADA algorithm introduced in Chapter 8. In the evaluation, SPADA was able to re-establish a homogeneous partitioning with regard to the AVPPs' mean predictability. (2) On the other hand, the evaluation investigated how the hierarchy changes over time (e.g., in terms of its height) with respect to different parametrizations of the dissolution and the introduction constraints.

In this thesis, we substantially extend this existing evaluation in several important aspects:

1. We show that the system is actually able to autonomously decompose the scheduling problem. Starting with a centralized system structure, the system is able to form a hierarchical structure that drastically reduces the time needed to create schedules.
2. We investigate different factors influencing the stability of the hierarchical structure. In this context, we demonstrate that the structure's stability not only hinges on the parametrization of the dissolution and the introduction constraints. In particular, its stability depends on the size and the number of the created partitions and the objective function used when solving the partitioning

problem. While the partition's size and number is steered by the partitioning constraints presented in Section 6.2, homogeneous partitioning again turns out to foster the structure's stability (cf. our results for flat system structures discussed in Section 6.3).

3. We examine the impact of the shape of the self-organizing structure and the above-mentioned factors on the quality of the created schedules in terms of the accuracy of the satisfaction of the demand and the resulting costs.
4. We show that a partitioning algorithm, such as PSOPP, is able to establish high-quality partitionings within the hierarchical structure.

Based on our power management case study, the results presented in this section show that the use of the partitioning constraints, the organizational paradigm of homogeneous partitioning, a partitioning algorithm like PSOPP or SPADA, and an adequate parametrization of the dissolution and introduction constraints lead to a self-organizing hierarchical system structure that comes to a compromise between stability, runtime performance, and the quality of solutions to the underlying resource allocation problem. Before we discuss our evaluation results, we introduce our test bed as well as the different examined configurations.

Test Bed

We perform the evaluation in our simulation environment for autonomous power systems. It is composed of 173 dispatchable and 350 non-dispatchable power plants of different types (hydro, biofuel, and gas power plants as well as solar plants and wind generators). Each power plant is modeled as an individual agent. The consumers are represented by a single agent. The demand, i.e., residual load, originates from weather-dependent power plants and the consumers. We use real world data for the capabilities of physical power plants¹ (such as production boundaries), the load curves², and the simulated weather conditions³ influencing the output of weather-dependent power plants. Each dispatchable power plant's inertia is defined within typical boundaries. The production costs, i.e., the unit price of electricity, range from $6.50 \frac{\text{euro cent}}{\text{kWh}}$ to $17.50 \frac{\text{euro cent}}{\text{kWh}}$. In each simulation run, power plants have to satisfy a prescribed residual load over a period of a full day, corresponding to 96 time steps, each representing 15 min. Every 15 min, AVPPs create schedules for the next hour on the basis of the predicted residual load. We also use a resolution of 15 min for the schedules so that each of them comprises 4 time steps. AVPPs create schedules according to the regio-central approach sketched in Section 2.2. As formalized in Equation (2.1), the AVPPs' primary goal is to create schedules that satisfy the predicted residual load as accurately as possible. The secondary goal is to minimize the costs. The optimization problems that have to be solved for creating the schedules are formulated as mixed integer linear programs and solved by the standard mathematical programming software IBM ILOG CPLEX⁴.

In each simulation run, we initialized the system with a centralized structure. As depicted in Figure 6.1a, the centralized structure consists of a single AVPP that controls all power plants. Besides creating schedules on the basis of the predicted residual load, the system's task was, starting from the centralized structure, to self-organize into an adequate hierarchy. The system's ability to self-organize was activated after the 16-th time step so that the system could adapt its structure in 80 of the overall 96 time steps per run. As we wanted to focus on vertical reorganizations in this evaluation (i.e., the dissolution and the introduction of AVPPs), we disabled the observation of the composition constraint in these experiments. With regard to the partitioning problem that has to be solved to introduce new AVPPs, the goal was to create a homogeneous partitioning in which the partitions' mean predictability (captured in the power plants' trust values) is as similar as possible. To solve the partitioning problem in response to the violation of the introduction constraint, each AVPP was equipped with an instance of the PSOPP algorithm (parameters were chosen according to the results of the parameter search

¹Energymap (Bavaria), 2012: <http://www.energymap.info>, retrieved in 2012.

²LEW, 2012: <http://www.lew-verteilnetz.de>, retrieved in 2012.

³LfL (Bavaria), 2010: <http://www.lfl.bayern.de/agm/>, retrieved in 2012.

⁴IBM ILOG CPLEX Optimizer, Version 12.4, 2011: <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, retrieved on March 1, 2016.

presented in Section 7.6). We opted for PSOPP in this evaluation as it turned out to be more suitable for one-dimensional partitioning problems than SPADA (see Section 8.5). While we evaluated different ranges for the size of partitions, we used a fixed number of partitions to reduce the size of the parameter space. We used $n_{min} = 5$ and $n_{max} = 523$ for the minimum and the maximum number of partitions, respectively. In all experiments, the threshold for the dissolution of an AVPP (i.e., the minimal allowed scheduling time) was set to 20 ms. To obtain the results summarized at the beginning of this section, we generated 33 different configurations. These stem from the different combinations of the following parametrizations:

1. To investigate the impact of the threshold of the introduction constraint, we used maximal allowed scheduling times of 60 ms, 120 ms, 240 ms, and 600 ms.
2. As for the influence of different ranges for the size of partitions, we used configurations with a minimum and maximum partition size of $s_{min} = 2$ and $s_{max} = 15$ (called “Small Partitions”) and another set of configurations with $s_{min} = 8$ and $s_{max} = 80$ (called “Large Partitions”).
3. To demonstrate that PSOPP is able to establish high-quality partitionings within the hierarchy and to investigate the influence of homogeneous partitioning on the system’s stability, its performance in terms of the maximum sequential scheduling time (i.e., the longest of all serial paths that result from adding the scheduling times for each branch in the hierarchy; see Equation (6.1)), its ability to satisfy the residual load, and the resulting costs, we used two different objective functions for the partitioning problem. In one half of the configurations, we maximize the fitness of homogeneous partitioning mean. In the other half, we do the opposite by minimizing the corresponding fitness. Where not stated otherwise, we regard configurations in which the fitness is maximized.
4. One half of the experiments was run in configurations in which a single violation of a dissolution or introduction constraint caused a vertical reorganization. The other half was based on configurations that employed the concept of *MaxSPAN* constraints introduced in [206]. In case of MaxSPAN constraints, the dissolution or introduction constraint only triggered a reorganization if the corresponding threshold was violated the second time within the last four time steps. Where not stated otherwise, we regard configurations using the MaxSPAN constraint.

To have a baseline for scheduling times and solution quality in terms of residual load satisfaction and costs, we used an additional configuration that prevented the system from changing its centralized structure in all 96 time steps. For each configuration, we performed 100 runs.

Results

In the following discussion of our evaluation results, we investigate the self-organizing hierarchy’s stability and its quality in terms of the AVPPs’ composition on the one hand, and its influence on the system’s scalability in terms of scheduling times and the quality of the created schedules on the other hand. Tables 9.1 and 9.2 present an overview of our evaluation results for the different configurations. We refer to these values in the course of our discussion.

Stability of the Hierarchical Structure and Quality of the Created Partitionings

As discussed in Section 6.3, a major concern is to establish a stable structure of AVPPs. When considering hierarchical system structures, stability not only depends on the need for horizontal reorganizations (which we investigated in Section 6.3), but also on the need for vertical reorganizations.

The evaluation provided by Steghöfer [206] pointed out that the stability of the hierarchy (measured by the number of reorganizations) can be increased by using MaxSPAN constraints instead of triggering a reorganization as soon as an AVPP’s scheduling time falls below or exceeds the predefined thresholds. We can reproduce these observations in our experiments using the AVPPs’ average lifetime as metric for stability.

Moreover, we expect the structure to be more stable, the larger the interval defined by the minimal and maximal (allowed) scheduling times. That is because fluctuations in the scheduling times leading to singular outliers do not immediately cause a violation of the dissolution or the introduction constraint.

Maximal Scheduling Time [ms]	Large Partitions				Small Partitions				Centralized SO Disabled
	60	120	240	600	60	120	240	600	
AVPP Lifetime per Run [ticks]	75.92 (2.25)	75.39 (2.55)	62.42 (9.08)	79.73 (3.92)	19.26 (1.66)	45.22 (5.84)	50.25 (5.35)	49.53 (6.15)	96.00 (0.00)
#Reorganizations per Run	9.61 (2.90)	8.91 (2.41)	4.87 (2.07)	1.08 (0.31)	433.56 (58.35)	58.73 (17.85)	43.53 (11.18)	43.83 (12.61)	0.00 (0.00)
Quality of Partitionings per Time Step	0.03 (0.02)	0.03 (0.02)	0.05 (0.03)	0.01 (0.01)	0.06 (0.02)	0.04 (0.02)	0.05 (0.02)	0.05 (0.02)	–
Hierarchy's Height per Time Step	2.97 (0.16)	2.98 (0.16)	2.89 (0.31)	2.00 (0.06)	6.55 (1.20)	4.10 (0.67)	3.35 (0.52)	3.17 (0.41)	1.00 (0.00)
#AVPPs per Time Step	47.71 (7.29)	46.29 (7.12)	23.50 (7.18)	11.72 (2.77)	110.44 (16.53)	59.72 (10.58)	54.59 (12.27)	52.96 (12.86)	1.00 (0.00)
#Children per AVPP and Time Step	12.71 (5.73)	13.01 (5.52)	25.61 (8.61)	47.66 (9.29)	5.83 (0.72)	9.97 (1.34)	10.98 (1.95)	11.35 (2.12)	523.00 (0.00)
Max. Sequential Scheduling Time per Time Step [s]	264.22 (61.25)	264.40 (62.35)	307.31 (78.83)	342.10 (75.46)	352.16 (69.50)	299.00 (70.19)	298.68 (63.95)	301.56 (67.11)	13134.50 (6839.08)
Scheduling Time per AVPP and Time Step [s]	56.65 (14.78)	57.28 (14.15)	85.66 (20.83)	137.16 (23.22)	39.30 (2.35)	50.05 (3.70)	52.59 (4.93)	53.43 (5.54)	13134.50 (6839.08)
Total Violation of the Assigned Residual Load per Time Step [kW]	10973.08 (11932.17)	11059.16 (13206.28)	1619.53 (4901.14)	175.76 (1581.84)	39335.36 (28958.82)	29464.34 (27712.99)	21880.10 (25262.69)	16723.61 (17501.21)	0.76 (54.96)
Violation of the Assigned Residual Load per AVPP and Time Step [kW]	222.51 (239.86)	230.67 (279.51)	61.65 (185.53)	16.07 (147.58)	353.32 (256.34)	487.85 (447.20)	381.63 (421.04)	300.47 (293.21)	0.76 (54.96)
Relative Violation of the Assigned Residual Load per AVPP and Time Step [%]	0.04 (0.18)	0.04 (0.30)	0.02 (0.10)	0.00 (0.02)	0.98 (26.45)	0.69 (49.14)	0.08 (0.55)	0.07 (0.26)	0.00 (0.00)
Total Costs of Schedules per Time Step [1000 €]	142.07 (43.25)	142.90 (42.82)	144.54 (42.31)	141.40 (43.32)	140.18 (43.39)	140.92 (43.15)	138.92 (43.93)	138.62 (43.74)	123.13 (48.47)

Table 9.1: Evaluation results for different combinations of the maximal scheduling times and the size of partitions (“Large Partitions”: $s_{min} = 8$ and $s_{max} = 80$; “Small Partitions”: $s_{min} = 2$ and $s_{max} = 15$) using MaxSPAN constraints as dissolution and introduction constraints. Agents aimed for homogeneous partitionings. All data is based on 100 runs, each comprising 96 time steps. Values in parentheses denote standard deviations. To avoid that an AVPP can decrease its costs by not covering its assigned residual load, we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$ for violated assigned residual loads.

With regard to a specific AVPP, such fluctuations occur because scheduling times depend on the concrete instance of the scheduling problem, which comprises the power plants’ current state as well as the residual load to distribute. Based on our findings in Section 6.3, we further hypothesize that homogeneous partitioning not only decreases the need for horizontal but also for vertical reorganizations and thus contributes to the hierarchy’s stability. The same applies to increasing the size of partitions, i.e., AVPPs. We justify this claim based on the fact that homogeneous partitioning – as applied in this evaluation – aims for AVPPs of similar mean predictability. Consequently, each AVPP consists of some well-predictable dispatchable power plants on the one hand, and some non-dispatchable power plants that are rather hard to predict on the other hand. The larger an AVPP, the lower the influence of a single power plant on its aggregated, i.e., collective, behavior, and thus the more stable the system’s structure.

Figure 9.5 confirms our expectations. Apart from the results for the runs without the MaxSPAN constraints, the figure is based on data provided in Tables 9.1 and 9.2). The runs without the MaxSPAN constraints achieved standard deviations of 5.65, 3.22, 12.83, and 10.94 time steps (ordered according to the “No-MaxSPAN” configurations shown in Figure 9.5). The highest average lifetime is achieved when employing MaxSPAN constraints, using a high maximal scheduling time, establishing homogeneous partitionings, and forming large partitions. Interestingly, the type of partitioning and the size of the partitions have a particularly significant effect on the system’s stability: Especially for lower maximal scheduling times, maximizing the partitionings’ fitness according to homogeneous partitioning mean causes much higher AVPP lifetimes than minimizing the fitness (cf. the “Min. Fitness” configurations). When minimizing the fitness, we obtain AVPPs that are either dominated by dispatchable or non-dispatchable power plants, which results in inhomogeneous scheduling times. This increases the need for dissolving existing or introducing new AVPPs and thus deteriorates the AVPPs’ lifetime.

Similar observations can be made with regard to the development of the mean number of reorganizations depicted in Figure 9.6. As can be seen, the agents begin to form a hierarchical system

Maximal Scheduling Time [ms]	Large Partitions (Minimize Fitness)				Small Partitions (Minimize Fitness)				Centralized SO Disabled
	60	120	240	600	60	120	240	600	
AVPP Lifetime per Run [ticks]	28.88 (3.35)	41.85 (3.71)	45.25 (5.42)	62.55 (5.25)	23.31 (1.65)	35.20 (4.01)	36.43 (4.35)	37.70 (3.44)	96.00 (0.00)
#Reorganizations per Run	77.43 (14.83)	33.23 (5.00)	16.59 (3.83)	4.35 (1.24)	334.95 (39.14)	133.66 (36.44)	119.62 (29.70)	107.75 (24.28)	0.00 (0.00)
Quality of Partitionings per Time Step	0.17 (0.09)	0.20 (0.09)	0.21 (0.09)	0.25 (0.08)	0.12 (0.05)	0.13 (0.06)	0.13 (0.06)	0.11 (0.07)	–
Hierarchy's Height per Time Step	5.19 (1.49)	3.12 (0.42)	2.94 (0.24)	2.01 (0.11)	7.87 (2.60)	4.76 (0.98)	3.95 (0.67)	3.13 (0.44)	1.00 (0.00)
#AVPPs per Time Step	34.58 (5.68)	28.02 (4.50)	18.70 (3.26)	11.15 (1.05)	114.96 (15.30)	84.27 (21.29)	79.16 (21.76)	73.85 (22.98)	1.00 (0.00)
#Children per AVPP and Time Step	16.75 (4.44)	20.21 (4.05)	29.81 (5.30)	48.16 (3.94)	5.62 (0.57)	7.53 (1.40)	8.01 (1.59)	8.63 (1.90)	523.00 (0.00)
Max. Sequential Scheduling Time per Time Step [s]	342.52 (73.16)	296.12 (66.85)	313.88 (84.06)	388.57 (91.62)	416.00 (94.20)	347.88 (91.20)	353.98 (86.14)	435.59 (98.10)	13134.50 (6839.08)
Scheduling Time per AVPP and Time Step [s]	66.13 (11.41)	75.06 (10.78)	96.35 (13.72)	140.17 (15.75)	37.95 (2.50)	43.36 (4.26)	44.67 (4.76)	45.22 (5.17)	13134.50 (6839.08)
Total Violation of the Assigned Residual Load per Time Step [kW]	36124.08 (26157.45)	28840.83 (24688.44)	6574.78 (12540.40)	99.40 (766.74)	43346.91 (45936.47)	29424.70 (23429.52)	22043.76 (19917.29)	13487.27 (16120.52)	0.76 (54.96)
Violation of the Assigned Residual Load per AVPP and Time Step [kW]	1031.57 (752.36)	1001.45 (842.97)	330.56 (635.39)	8.76 (67.60)	377.83 (387.56)	357.61 (285.52)	275.50 (237.88)	174.94 (207.56)	0.76 (54.96)
Relative Violation of the Assigned Residual Load per AVPP and Time Step [%]	0.13 (0.30)	0.26 (14.50)	0.03 (0.09)	0.00 (0.04)	4.78 (209.28)	0.21 (4.56)	0.11 (0.33)	0.17 (6.27)	0.00 (0.00)
Total Costs of Schedules per Time Step [1000 €]	144.07 (42.32)	141.33 (43.67)	144.11 (42.54)	140.67 (43.91)	141.08 (42.17)	139.74 (42.60)	137.04 (43.16)	130.96 (45.92)	123.13 (48.47)

Table 9.2: Evaluation results for different combinations of the maximal scheduling times and the size of partitions (“Large Partitions”: $s_{min} = 8$ and $s_{max} = 80$; “Small Partitions”: $s_{min} = 2$ and $s_{max} = 15$) using MaxSPAN constraints as dissolution and introduction constraints. In contrast to Table 9.1, agents aimed for partitionings that *minimize* the fitness of homogeneous partitioning. All data is based on 100 runs, each comprising 96 time steps. Values in parentheses denote standard deviations. To avoid that an AVPP can decrease its costs by not covering its assigned residual load, we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$ for violated assigned residual loads.

structure as soon as their ability to self-organize is enabled after the 16-th time step. This is indicated by the initial peak of the number of reorganizations between time steps 17 and 25 (depending on the chosen configuration). Although this peak is followed by a decline in the number of reorganizations for all depicted configurations, the structure stabilizes best when forming large AVPPs, establishing homogeneous partitionings, and using a relatively high threshold for the maximal scheduling time. With respect to Figure 9.6, the hierarchical structure stabilizes if the number of reorganizations converges to 0. As is the case with the AVPPs’ lifetime, we observe that the structure’s stability hinges on the characteristics of the created partitionings in particular.

When taking a closer look at the development of the hierarchy’s height (see Figure 9.7a), we notice that, as expected, the hierarchy remains the flatter, the higher the maximal scheduling time. We further observe that all curves converge to a specific value (the rate and the actual value depend on the maximal scheduling time). This suggests that the structure stabilizes for maximal scheduling times of 60 ms, 240 ms, and 600 ms. However, as the development of the number of AVPPs shows (see Figure 9.7b), this is actually only the case for 600 ms (note that the number of AVPPs can be changed without modifying the hierarchy’s height). In case of a low threshold of 60 ms, the system creates more or less instantaneously a deep hierarchy comprising a large number of small AVPPs. As some of these AVPPs dissolve over time, it seems that the agents overreact during the first time steps in which they can reorganize the structure. Using a threshold of 240 ms lowers the rate of convergence of the hierarchy’s height. The system keeps on creating more and more AVPPs. The threshold of 600 ms seems to be most suitable in terms of stability. While the structure remains relatively flat on average, the number of AVPPs stabilizes quickly; only slight adjustments are needed over time. Evidently, using tighter bounds for the partitions’ maximum size results in deeper hierarchies and more AVPPs (cf. Tables 9.1 and 9.2).

As indicated by the different values for the quality of the created partitionings when minimizing or maximizing the fitness, the hierarchical structure does not prevent a partitioning algorithm, such

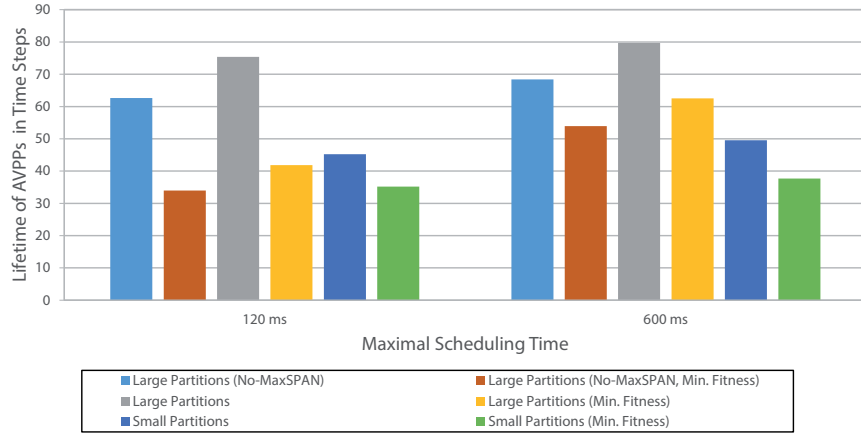


Figure 9.5: Mean lifetime of AVPPs in time steps for different combinations of maximal scheduling times, the size of partitions, the employed objective function, and the type of dissolution and introduction constraints. Where not stated otherwise (see “No-MaxSPAN” and “Min. Fitness”), AVPPs employed MaxSPAN constraints as dissolution and introduction constraints, and agents aimed for homogeneous partitionings. The AVPPs’ lifetime is improved by using higher thresholds for the maximal scheduling time, increasing the size of partitions, and establishing homogeneous partitionings. The maximal possible lifetime amounts to 96 time steps.

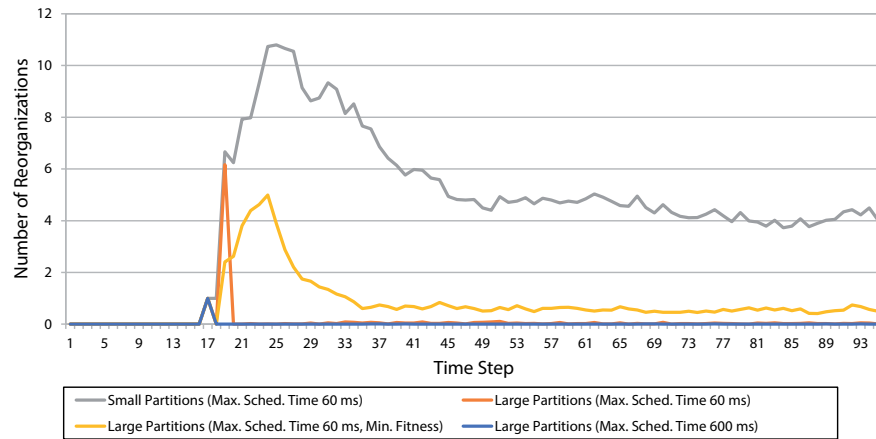


Figure 9.6: Development of the mean number of reorganizations performed over time. The agents’ ability to self-organize is enabled after time step 16. For all configurations, results are averaged over 100 runs. The lower the number of reorganizations, the more stable the system structure.

as PSOPP or SPADA, from finding partitionings that comply with the employed objective function (the following data are aggregates of those listed in Tables 9.1 and 9.2): When maximizing the fitness, PSOPP obtains mean qualities of 0.03 (standard deviation $\sigma = 0.01$) and 0.05 ($\sigma = 0.01$) for large and small partitions, respectively (the lower the value for the mean quality, the better). Otherwise, when minimizing the fitness, PSOPP yields mean qualities of 0.21 ($\sigma = 0.03$) and 0.12 ($\sigma = 0.01$) for large and small partitions, respectively (the higher the value for the mean quality, the better).

To summarize these insights, despite the hierarchical structure, it is very important to *optimize* the AVPPs’ composition according to a specific objective function and to control the size and the number of created subsystems by means of the partitioning constraints. Taking this into account, the agents are able to reorganize the initially centralized system structure into a stable hierarchical structure.

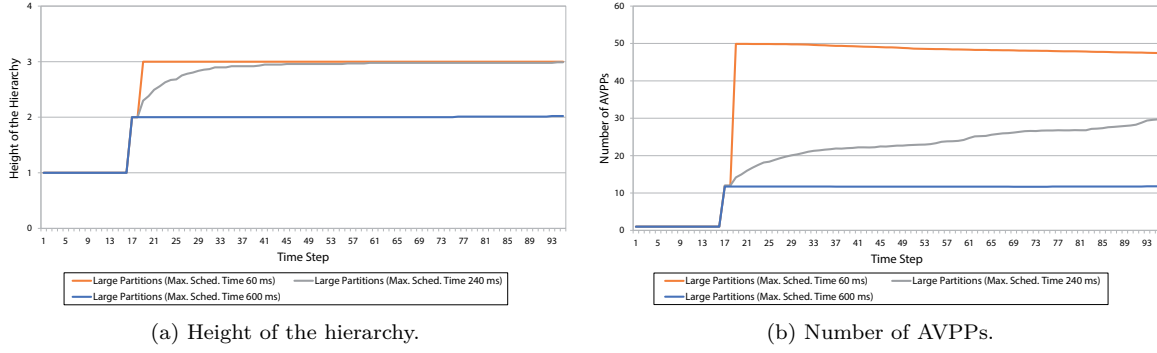


Figure 9.7: Development of the mean height of the hierarchy (see Figure 9.7a) and the mean number of AVPPs (see Figure 9.7b) over time. The agents' ability to self-organize is enabled after time step 16. For all configurations, results are averaged over 100 runs. The initially centralized system structure has a height of one and consists of a single AVPP.

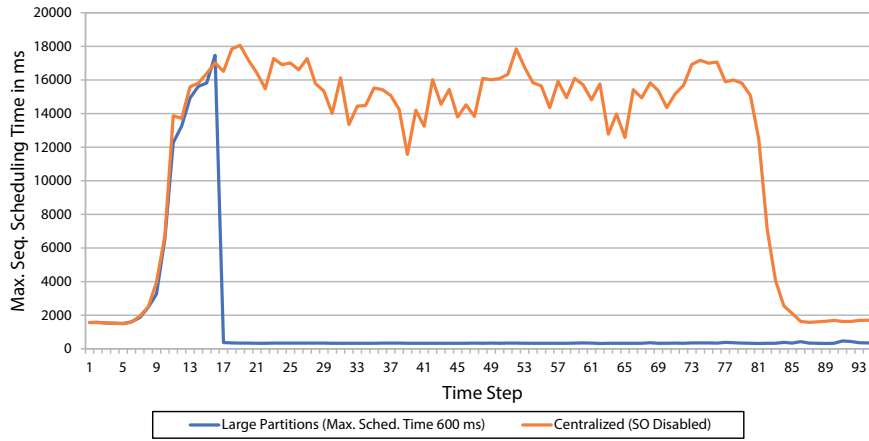


Figure 9.8: Development of the maximum sequential scheduling time for a centralized system without the capability to self-organize, compared to a configuration in which the agents' ability to self-organize is enabled after the 16-th time step. For both configurations, results are averaged over 100 runs. The development of the scheduling times of the centralized system reveals that the complexity of solving the scheduling problem varies with the time of day.

Scheduling Times

With regard to scheduling times, we suppose that the self-organized formation of a hierarchical system structure significantly reduces the maximum sequential scheduling time needed for schedule creation in a centralized system. Figure 9.8 confirms that this is indeed the case. In time step 17 – right after enabling the system's ability to self-organize –, the self-organizing system obtains a max. seq. scheduling time of only 371.15 ms ($\sigma = 40.40$ ms), compared to 16512.83 ms ($\sigma = 3987.14$ ms) in case of the centralized system. Since the coefficient of variation decreases from $\frac{3987.14}{16512.83} \approx 0.24$ to $\frac{40.40}{371.15} \approx 0.11$, the self-organizing hierarchy also provides much more stable scheduling times. On average, the max. seq. scheduling time is reduced by 97.40 % when using a threshold of 600 ms for the introduction constraint. Interestingly, as shown by the scheduling times of the centralized system, the complexity of solving the scheduling problem varies with the time of day. As stated before, the scheduling problem's complexity depends on different factors, including the power plants' state and the residual load to distribute. This highlights the need for a self-organizing system structure that can adapt in response to changes in the

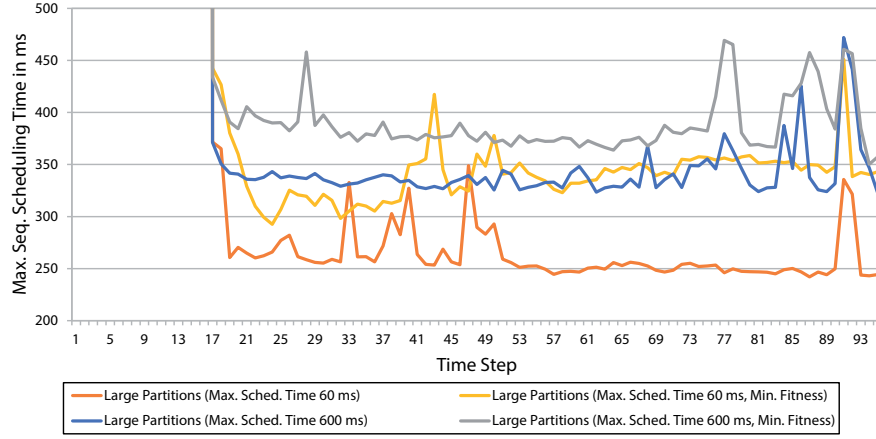


Figure 9.9: Comparison of the development of the maximum sequential scheduling time for different configurations in which the agents’ ability to self-organize is enabled after the 16-th time step. For all configurations, results are averaged over 100 runs.

environmental conditions and the agents’ internal states.

With respect to the self-organizing hierarchy, we hypothesize that not only the maximal (allowed) scheduling time and thus the hierarchy’s height, the AVPPs’ size, as well as their number have an influence on the max. seq. scheduling time, but also the objective function used for optimizing the AVPPs’ composition. Concerning the latter, we expect that the use of homogeneous partitioning reduces the max. seq. scheduling time because it favors AVPPs featuring an optimized balance of dispatchable and non-dispatchable power plants. That way, homogeneous partitioning “distributes” the complexity of schedule creation evenly among the AVPPs, which results in a shorter max. seq. path with regard to the scheduling times. Otherwise, when minimizing the fitness, AVPPs are more dissimilar so that one AVPP is likely to need more time for creating schedules than another, which is why the max. seq. scheduling time is not efficiently reduced.

Figure 9.9 shows the hypothesized behavior for configurations in which we prescribe the creation of large partitions. Compared to a threshold of 600 ms for the maximal scheduling time, a threshold of 60 ms causes significantly smaller AVPPs (see Figure 9.7b). As a consequence, the lower threshold yields shorter max. seq. scheduling times (on average, 264.22 ms with $\sigma = 61.25$ ms compared to 342.10 ms with $\sigma = 75.46$ ms when maximizing the fitness, and 342.52 ms with $\sigma = 73.16$ ms compared to 388.57 ms with $\sigma = 91.62$ ms when minimizing the fitness). As for the influence of homogeneous partitioning, the configuration “Large Partitions (Max. Sched. Time 600 ms)” can obtain similar max. seq. scheduling times as the configuration “Large Partitions (Max. Sched. Time 60 ms, Min. Fitness)”. For all combinations of maximal scheduling times and prescribed ranges for the size of partitions, the max. seq. scheduling time is higher when minimizing the fitness, i.e., creating dissimilar AVPPs (see Tables 9.1 and 9.2).

The data for the mean scheduling time per AVPP additionally support our hypothesis. The negative impact of dissimilar AVPPs becomes evident when comparing Figure 9.10 with Figure 9.9: Figure 9.10 shows that the mean scheduling time per AVPP is much shorter when using the lower threshold of 60 ms, which results in smaller AVPPs. The fact that the configuration “Large Partitions (Max. Sched. Time 600 ms)” obtains approximately the same max. seq. scheduling time as the configuration “Large Partitions (Max. Sched. Time 60 ms, Min. Fitness)” points out that a structure consisting of similar AVPPs should be preferred to one comprising dissimilar AVPPs.

The data listed in Tables 9.1 and 9.2 show that deeper hierarchies consisting of a higher number of smaller AVPPs do not necessarily lead to shorter max. seq. scheduling times. In case of “Large Partitions”, we actually reach the shortest average max. seq. scheduling time of 264.22 ms ($\sigma = 61.25$ ms) with a threshold of 60 ms, which causes relatively deep hierarchies (average height of 2.97 with $\sigma = 0.16$) consisting of relatively small AVPPs (12.71 children per AVPP on average, with $\sigma = 5.73$). At first

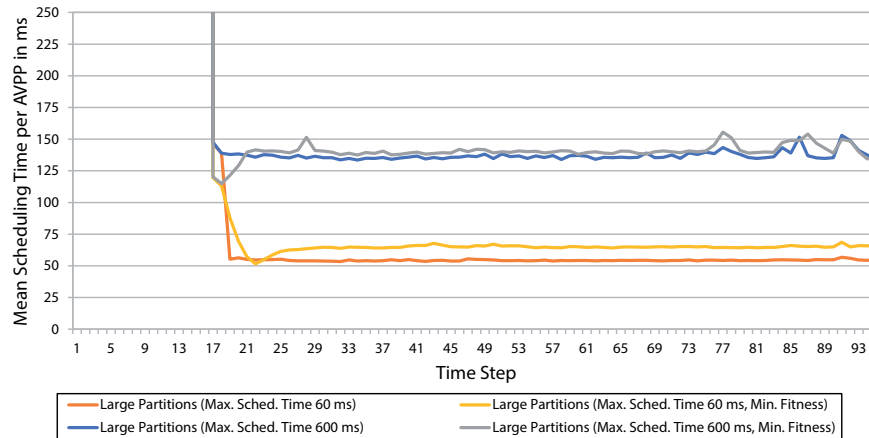


Figure 9.10: Comparison of the development of the mean scheduling time per AVPP for different configurations in which the agents’ ability to self-organize is enabled after the 16-th time step. For all configurations, results are averaged over 100 runs. Regarding the average of the mean scheduling time per AVPP over all time steps, we obtain 56.65 ms ($\sigma = 14.78$ ms) for “Large Partitions (Max. Sched. Time 60 ms)”, 66.13 ms ($\sigma = 11.41$ ms) for “Large Partitions (Max. Sched. Time 60 ms, Min. Fitness)”, 137.16 ms ($\sigma = 23.22$ ms) for “Large Partitions (Max. Sched. Time 600 ms)”, and 140.17 ms ($\sigma = 15.75$ ms) for “Large Partitions (Max. Sched. Time 600 ms, Min. Fitness)”.

glance, this seems to contradict our statement. However, this scheduling time is shorter than the 352.16 ms ($\sigma = 69.50$ ms) obtained with the same threshold when demanding the formation of small partitions. Here, the hierarchy’s average height is 6.55 ($\sigma = 1.20$) and AVPPs have only 5.83 children on average ($\sigma = 0.72$). In case of “Small Partitions”, a threshold of 240 ms yields the shortest max. seq. scheduling time of 298.68 ms on average ($\sigma = 63.95$ ms), based on hierarchies with a mean height of 3.35 ($\sigma = 0.52$) and an average of 10.98 children per AVPP ($\sigma = 1.95$). Even the max. seq. scheduling time obtained with a threshold of 600 ms is shorter than the one obtained with a threshold of 60 ms. With regard to “Large Partitions (Minimize Fitness)” and “Small Partitions (Minimize Fitness)”, a threshold of 120 ms yields the shortest max. seq. scheduling times of 296.12 ms ($\sigma = 66.85$ ms) and 347.88 ms ($\sigma = 91.20$ ms), respectively. Again, the call for large partitions results in shorter max. seq. scheduling times. Note that, in all cases, the mean scheduling time per AVPP increases with the threshold for the maximal scheduling time, though. The higher the maximal scheduling time, the larger the AVPPs and the flatter the hierarchy.

On the one hand, these results demonstrate the need for self-organizing system structures that allow the agents to adapt to changes in their internal states and their environment. On the other hand, they show that a partitioning algorithm (in this case PSOPP), combined with the Hierarchical Control Loop, is able to autonomously form an adequate hierarchical structure at runtime. Further, our data confirmed the advantage of creating hierarchical structures on the basis of homogeneous partitioning and the partitioning constraints since a hierarchy of appropriate height consisting of similar AVPPs of adequate size and number significantly increases the system’s runtime performance.

Quality of Created Schedules

Based on the observations we made when creating schedules in static hierarchies (cf. the short summary at the beginning of Chapter 6), we expect that the hierarchy’s height as well as the size and the number of AVPPs have a significant influence on the schedules’ quality in terms of the violation of the assigned residual load and the costs. While deep hierarchies consisting of small AVPPs can decrease the max. seq. scheduling time (as discussed above, there are sweet spots), they come at the expense of solution quality due to fragmentation. Fragmentation gives rise to abstraction errors which cause AVPPs to under- or overestimate the capabilities of subordinate AVPPs when creating schedules. Underestimations typically

lead to false or suboptimal decisions that directly worsen the schedules' quality (e.g., underestimating an AVPP's price-performance ratio can increase the costs of energy provision). Overestimations can be cushioned if the affected AVPP features a sufficient amount of degrees of freedom (depending on the controllability of its dispatchable power plants) and redundancy (e.g., multiple similar low-priced power plants). Increasing fragmentation intensifies the negative impact of overestimations because it reduces the AVPPs' degrees of freedom and redundancy. Hence, there is a trade-off between the max. seq. scheduling time and the schedules' quality. The data provided in Tables 9.1 and 9.2 capture this trade-off.

The negative consequences of fragmentation become especially visible when comparing the results for maximal scheduling time thresholds of 60 ms and 600 ms. For "Large Partitions" (see Table 9.1), for instance, we obtain an average total violation⁵ of the assigned residual load of 10973.08 kW ($\sigma = 11932.17$ kW) and 175.76 kW ($\sigma = 1581.84$ kW), respectively. So the violation resulting from the 600 ms threshold is only 1.60 % of the violation caused by the fragmented structure when using a 60 ms threshold. On the other hand, the max. seq. scheduling times show only minor differences (on average, 264.22 ms with $\sigma = 61.25$ ms for the 60 ms threshold compared to 342.10 ms with $\sigma = 75.46$ ms for the 600 ms threshold). Note that the violation of the centralized system is only 0.76 kW ($\sigma = 54.96$ kW). Similar observations can be made with regard to the mean violation of the assigned residual load per schedule creation: Here, we obtain mean violations of 222.51 kW ($\sigma = 239.86$ kW) and 16.07 kW ($\sigma = 147.58$ kW), respectively. Fortunately, the mean relative violations per schedule creation of 0.04 % ($\sigma = 0.18$ %) for 60 ms and 0.00 % ($\sigma = 0.02$ %) for 600 ms remain low for both thresholds. So despite significant differences in the absolute numbers of the total violation of the assigned residual load and of the mean violation of the assigned residual load per schedule creation, the residual load was sufficiently satisfied in both configurations. For "Large Partitions", the mean total costs only show slight differences for different thresholds (the costs for the 60 ms threshold are, on average, 0.47 % higher than those for the 600 ms threshold) but are approximately 15 % higher than those obtained by the centralized approach. The lower costs obtained by the higher threshold can be attributed to the flatter hierarchy (mean height of 2.00 with $\sigma = 0.06$ compared to 2.97 with $\sigma = 0.16$). We deduce this from the more considerable increase in the total costs with the hierarchy's height when minimizing the fitness (see Table 9.2), e.g., 14.58 % in case of "Small Partitions (Minimize Fitness)".

Apart from the threshold used for the introduction of new AVPPs, also the employed objective function of the partitioning problem and the parametrization of the partitioning constraints effect the system's fragmentation (see Tables 9.1 and 9.2): When maximizing the fitness of homogeneous partitioning, the mean total violation of the assigned residual load is 5956.89 kW ($\sigma = 5085.01$ kW) and 26850.86 kW ($\sigma = 8514.15$ kW) for large and small partitions, respectively. By contrast, when minimizing the fitness, the mean violation sums up to 17909.77 kW ($\sigma = 14974.48$ kW) and 27075.66 kW ($\sigma = 10957.14$ kW) for large and small partitions, respectively. Similar results are obtained for the mean violation of the assigned residual load per schedule creation. Regarding the relative violation per schedule creation, we obtain means of 0.03 % ($\sigma = 0.02$ %) for large partitions and 0.46 % ($\sigma = 0.39$ %) for small partitions when maximizing the fitness. When minimizing the fitness, the mean relative violations increase to 0.11 % ($\sigma = 0.10$ %) and 1.32 % ($\sigma = 2.00$ %), respectively. In tune with our previous observations, homogeneous partitioning and a parametrization of the partitioning constraints that allows for the creation of larger partitions increase the system's ability to achieve its goal of satisfying the residual load. Interestingly, we make contrary observations with respect to the mean total costs (see Tables 9.1 and 9.2). Here, small partitions – residing in a relatively flat hierarchy that reduces the probability of abstraction errors – yield the lowest costs (cf. "Small Partitions (Minimize Fitness)" in combination with a threshold of 600 ms). We ascribe this to the fact that, using a fixed number of sampling points, the cost functions of smaller AVPPs are more accurate than those of large AVPPs. We suppose that we could decrease the costs caused by large AVPPs if we increased the number of sampling points. We leave the analysis of this phenomenon for future work.

⁵Note that we concentrate on the AVPPs' ability to redistribute the assigned residual load instead of measuring the deviation between actual production and consumption in this evaluation.

Chapter Summary and Outlook

In this chapter, we presented the Hierarchical Control Loop that is run on each intermediary in the hierarchy to regulate the self-organized formation of hierarchical system structure. To this end, the Hierarchical Control Loop associates three elementary constraints – the dissolution, the introduction, and the composition constraint – with a constraint-specific control action that is triggered as soon as a violation of the associated constraint is observed. If the scheduling time falls below (violation of the dissolution constraint) or exceeds (violation of the introduction constraint) a predefined threshold a vertical reorganization is triggered that dissolves the initiating intermediary or increases the degree of problem decomposition by introducing a new set of intermediaries. If a violation of the composition constraint is monitored, the corresponding intermediary triggers a horizontal reorganization that re-establishes an adequate partitioning in terms of the application-specific formation criteria. At its heart, the Hierarchical Control Loop employs a partitioning algorithm, such as PSOPP or SPADA, because horizontal reorganizations as well as the introduction of new intermediaries require solving instances of the partitioning problem.

Our evaluation confirmed that, in conjunction with the Hierarchical Control Loop, a partitioning algorithm, such as PSOPP, enables a system to self-organize into a structure that significantly reduces the time needed to create schedules. Based on suitable thresholds for the dissolution and introduction of AVPPs, the system is able to maintain an adequate quality of the created schedules. We expect that the formation of the hierarchy scales up to arbitrary large systems because reorganizations only affect specific subsystems whose sizes can be controlled by the partitioning constraints. While this reduces the complexity of the partitioning problems to be solved, PSOPP's and SPADA's ability to partition large sets of agents additionally support our hypothesis (see Sections 7.6 and 8.5). Throughout our evaluation, homogeneous partitioning as well as the ability to steer the degree of problem decomposition by means of the partitioning constraints proved to be beneficial to increase (1) the structures' stability, (2) the system's performance in terms of scheduling times, and (3) the schedules' quality. These observations are in line with those we made for flat system structures of height 2 in Section 6.3.

In future work, we want to apply this principle of autonomous problem decomposition to other application domains. Our evaluation showed that the quality of a structure depends heavily on the thresholds for the scheduling times and an appropriate parametrization of the partitioning constraints. Because it is not feasible to specify a set of suitable parameters for all possible situations at design time, new techniques have to be devised that enable the system to find these parameters itself at runtime. Such parameters yield structures that allow for short scheduling times and high-quality schedules. Given that scheduling times and solution quality depend on various factors, such as the composition of the subsystems within the hierarchy, the agents' current state, and the predicted demand, a solution to this problem has to cope with a very large search space.

The following part tackles the challenge of creating robust solutions to the resource allocation problem on the basis of the hierarchical system structures introduced in this chapter.

Part IV

Robust Solutions to the Resource Allocation Problem in Self-Organizing Hierarchies

In this thesis, we regard a resource allocation problem that has to be solved by a group of dispatchable agents whose behavior is subject to inertia. For this reason, they have to create schedules on the basis of demand predictions for a specific time span in advance. However, solving the scheduling problem is NP-hard with regard to the number of dispatchable agents involved and time steps schedules have to be created for. Furthermore, the agents have to balance supply and demand despite inaccurate demand predictions and dispatchable agents that struggle to contribute as scheduled.

In this part, we introduce techniques for finding robust solutions to the resource allocation problem in self-organizing hierarchies. Chapter 10 outlines an auction- and trust-based mechanism, called *TruCAOS*, for scalable schedule creation in large-scale systems. Among other trust-based measures, it implements a risk-avoidance strategy that mitigates uncertainties originating from schedule violations. In Chapter 11, we borrow and adapt techniques from the field of *online stochastic optimization* [83] to create schedules that enable the system to deal with several possible developments of volatile demand that follows different behavioral patterns. To this end, we apply the concept of *Trust-Based Scenario Trees* (TBSTs) presented in Chapter 5 to hierarchical schedule creation. Together with the ability to choose the most suitable scenario at runtime, TBST-based schedules avoid situations in which deviations between supply and demand are higher than (technically) allowed, or where their compensation is either very costly or not feasible due to the dispatchable agents' inert behavior. To improve the dispatchable agents' ability to deal with unforeseen situations, we further allow them to schedule an appropriate amount of degrees of freedom, i.e., *reserves*. Although the demand has to be satisfied in a fine-grained time pattern, the scheduling problem's complexity and the uncertainties involved call for rather coarse-grained schedules. To maintain the balance between supply and demand at all times, the dispatchable agents have to reactively adapt their supply in compliance with the actual demand and their scheduled contribution. In Chapter 12, we show that TBST-based schedules *proactively guide* the dispatchable agents' *permanent reactive supply adjustments*. To live up to the strongly connected nature of the addressed issues, Chapter 13 provides a cumulative discussion of the related work. Nevertheless, we still refer to related work in the other chapters where appropriate.

TruCAOS – an Auction- and Trust-Based Mechanism for Scalable Schedule Creation

Summary. In this chapter, we introduce an auction-based mechanism, called *TruCAOS*, that solves the scheduling problem in hierarchical systems in a cooperative and regionalized manner. In contrast to the regio-central approach (see Section 2.2) which accumulates the complexity of solving the scheduling problem in intermediaries, TruCAOS enables an intermediary’s dispatchable agents to actively participate in the process of schedule creation. Because this reduces the complexity of schedule creation, the agents can self-organize into larger subsystems, which results in flatter and more stable hierarchies, prevents the introduction of unnecessary abstraction errors, and is thus advantageous to the schedules’ quality. These characteristics qualify TruCAOS for scalable schedule creation in large-scale systems. To cope with uncertain supply and demand, TruCAOS implements a variety of trust-based measures. In particular, TruCAOS actively reduces uncertainties stemming from schedule violations by “moving” the dispatchable agents’ supply into regions in which their behavior is more predictable. In case of avoidable misbehavior, a payment function that distributes rewards according to the agents’ actual instead of their scheduled supply and that penalizes schedule violations incentivizes compliance with schedules. TruCAOS further uses trust values to quantify and anticipate systematic deviations from demand predictions. Our evaluation confirms that these characteristics allow TruCAOS to create high-quality schedules in much shorter time than the regio-central approach. We also demonstrate that TruCAOS’s trust-based measures reduce the impact of inaccurate demand predictions and the presence of aleatoric uncertainties originating from schedule violations.

Publication. The concepts and results outlined in this chapter have been published in Anders et al. [10, 14].

As demonstrated in Section 9.2, self-organizing hierarchical system structures significantly reduce the time needed to create schedules. However, decreasing scheduling times by introducing new hierarchy levels increases the system’s fragmentation and thus usually comes at the expense of solution quality in terms of demand satisfaction and costs. A fragmented system is mainly characterized by small subsystems that form a relatively deep hierarchy. This gives rise to abstraction errors that cause intermediaries to under- or overestimate the capabilities of subordinate intermediaries when creating schedules. Aside from that, the environment’s dynamic and uncertain nature requires to revise the schedules frequently (see Section 1.3). It is therefore more beneficial to be able to update schedules frequently with a fast but possibly sub-optimal method than spending much time on calculating *optimal* schedules for a problem whose conditions might change during calculation. In other words, sparing no effort in finding an optimal solution to the scheduling problem would be out of proportion to the benefit. This not only justifies our approach to decompose the scheduling problem by means of a self-organizing hierarchical system structure as shown in Part III, but also the application of heuristics for solving the

scheduling problem within each organization (i.e., AVPP).

Interestingly, fast scheduling algorithms do not only allow the system to update schedules frequently, but also to solve the scheduling problem for a large group of dispatchable agents in short time. In comparison to less efficient scheduling algorithms, agents can thus self-organize into flatter hierarchies consisting of larger subsystems. Because such structures are more stable and less susceptible to abstraction errors (see Section 9.2), fast scheduling algorithms lay the foundation for high-quality schedules in terms of demand satisfaction and costs.

In this chapter, we introduce an auction-based algorithm, called *TruCAOS*¹, that solves the scheduling problem in a cooperative and regionalized manner. As the provision and consumption of resources, such as electricity, are already subject to rewards and costs in real systems, a market-based approach is a natural choice [220]. Together with the principle of self-organized hierarchical problem decomposition, TruCAOS enables the creation of high-quality schedules in large-scale systems.

Just as the regio-central approach (see Section 2.2), TruCAOS creates schedules in a top-down manner. But instead of accumulating the complexity of schedule creation in the intermediaries and *assigning* schedules to subordinates, TruCAOS enables an intermediary’s dispatchable agents to actively participate in the schedule creation. When creating schedules, an intermediary merely acts in the role of an auctioneer that redistributes its fraction of the overall demand in an iterative and incremental process. In each iteration, subordinate dispatchable agents generate proposals that contain an agent’s updated schedule. The intermediary identifies and accepts a combination of proposals that increases the satisfaction of the demand at low costs. Because these auctions are limited to an intermediary’s directly subordinate agents, we refer to TruCAOS as a *regionalized* approach.

Moreover, TruCAOS implements a variety of trust-based measures that allow for the creation of adequate schedules despite inaccurate demand predictions and dispatchable agents that struggle to contribute as promised. Here, we distinguish between deviations from contracts that originate from *unavoidable* and *avoidable* misbehavior. Unavoidable misbehavior comprises deviations that result from, e.g., inaccurate consumption or production predictions due to outdated standard load profiles or inaccurate sensors. Avoidable misbehavior refers to deviations that could have been prevented. For instance, the owner of a biogas power plant could ensure that the plant is in a proper condition, which prevents schedule violations. Intentional misbehavior (e.g., lying) can always be classified as avoidable misbehavior because it can be avoided by an adequate incentive mechanism (e.g., the owner of a biogas power plant is incentivized that its plant is able to comply with its schedule). By contrast, it is often not possible to avoid unintentional misbehavior. An agent cannot be incentivized to behave benevolently if it is not in its power to change its behavior, e.g., if its behavior is governed by environmental conditions, such as the weather. In particular, we presume that imprecise demand predictions always result from unavoidable misbehavior. As for the dispatchable agents’ supply, we assume that schedule violations can stem from both unavoidable as well as avoidable misbehavior. For this reason, we differentiate between a dispatchable agent’s (internal and private) *scheduled* supply, its *promised* supply communicated in the form of a proposal, and its *actual* supply. The scheduled supply is the supply an agent is actually *willing* to provide. If we may assume that agents do not make false reports about their scheduled supply, scheduled and promised supply do not have to be distinguished. While we primarily focus on methods for dealing with unavoidable misbehavior, we do not exclude avoidable misbehavior from our investigations in this chapter.

In general, TruCAOS employs trust models to (1) anticipate aleatoric, i.e., intrinsic random and irreducible, uncertainties, (2) decrease epistemic, i.e., systematic, uncertainties, and (3) reduce the *presence* of aleatoric uncertainties. While TruCAOS uses trust values to alleviate epistemic uncertainties in demand predictions, we concentrate on uncertain supply in this chapter. To cope with uncertain supply, each intermediary assesses the behavior of each of its subordinate dispatchable agents by means of the two trust values presented in Section 4.2: The first trust value records a dispatchable agent’s *mean* deviation from its promised supply and thus captures systematic misbehavior. The second trust value is used to mirror a dispatchable agent’s *predictability* and thus the risk that the goal of balancing supply and demand is not achieved because the agent does not provide resources as promised or expected.

¹“TruCAOS” stands for *trust- and cooperation-based algorithm for open MAS*.

Both trust values are assessed as a function of an agent’s scheduled supply. Intermediaries incorporate these trust values when selecting winner proposals to anticipate uncertainties and incentivize benevolent behavior: Instead of relying on a proposed supply, intermediaries use a dispatchable agent’s first trust value to determine its expected supply, thereby reducing epistemic uncertainties. Furthermore, the algorithm makes decisions that favor predictable agents. These “move” the dispatchable agents’ supply into regions in which their behavior is more predictable, which reduces aleatoric uncertainties. This can be thought of as preferring a biased, thus more predictable, coin favoring tail with 90 % to a perfectly fair, hence unpredictable, coin. In conjunction with a payment function that distributes rewards according to the agents’ actual instead of their scheduled supply and that penalizes schedule violations, these measures do not only lower the influence of unavoidable misbehavior but also incentivize benevolent behavior.

In Section 10.1, we explain how a hierarchy of dispatchable agents creates schedules by means of TruCAOS. Section 10.2 outlines the intermediaries’ procedure for identifying an appropriate set of proposals that should be accepted. In its simplest form, TruCAOS does not include any measures of prediction accuracy and does not allow intermediaries to identify dispatchable agents that make erroneous proposals. In Section 10.3, we therefore present two trust metrics TruCAOS employs to quantify and anticipate deviations between predicted and actual demand, and promised and actual supply. These trust metrics are build on the basic trust model defined in Section 4.2. Based on these trust metrics, we introduce TruCAOS’s trust-based measures to deal with uncertain demand and supply in Section 10.4. In Section 10.5, we demonstrate that these measures mitigate the influence of imprecise demand predictions and can actively reduce deviations from promised supply. Furthermore, we show that TruCAOS creates high-quality schedules in much shorter time than the regio-central approach.

We discuss related work from the field of trust-based resource allocation in Chapter 13. In Chapter 11, we enhance TruCAOS in two ways that improve the system’s robustness with regard to uncertain demand: First, we enable TruCAOS to create schedules for several possible developments of volatile demand that follows different behavioral patterns on the basis of *Trust-Based Scenario Trees* (see Chapter 5). Second, we allow dispatchable agents to schedule *reserves*, i.e., an adequate amount of degrees of freedom. These promote the agents’ ability to reactively adjust their supply according to the actual demand.

10.1 Basic Procedure of TruCAOS

Similarly to the regio-central approach sketched in Section 2.2, TruCAOS determines schedules in a top-down manner. The schedule creation is triggered by the top-level intermediary Λ by retrieving an aggregated demand prediction $\hat{D}^{\mathcal{W}}$ for all N time steps in the scheduling window $\mathcal{W} = \{t_{\text{now}} + i \cdot \Delta\tau \leq t_{\text{now}} + H \mid i \in \mathbb{N}_{\geq 1}\}$ from all non-dispatchable agents (in our case study, non-dispatchable consumers and weather-dependent power plants). Subordinate intermediaries $\lambda \in \mathcal{I} \setminus \{\Lambda\}$ are responsible for recursively distributing their fraction of the overall demand, that is, their **scheduled supply** S_λ . In contrast to the regio-central approach, which *assigns* schedules, TruCAOS does not accumulate the complexity of solving the scheduling problem in the intermediaries, though. To decrease complexity, each intermediary’s subordinate dispatchable agents \mathcal{V}_λ become an active part in the scheduling process by enabling them to sell or buy resources according to the demand it has to distribute. This is done in an iterative and incremental process that, in its basic form, is reminiscent of an iteratively performed first-price sealed-bid auction in which intermediaries act as auctioneers (see, e.g., [119]). Figure 10.1 gives an overview of the single steps this process comprises.

Since the scheduled supply S_a of each dispatchable agent $a \in \mathcal{V}$ must be feasible with respect to its control model (see Equation (1.3)), all intermediaries $\lambda \in \mathcal{I}$ ask their subordinate dispatchable agents \mathcal{V}_λ to perform and communicate so-called *schedule corrections* before the iterative and incremental distribution of the demand is started. We elaborate on why this is necessary and on how this step is accomplished in the course of the following more detailed description of TruCAOS.

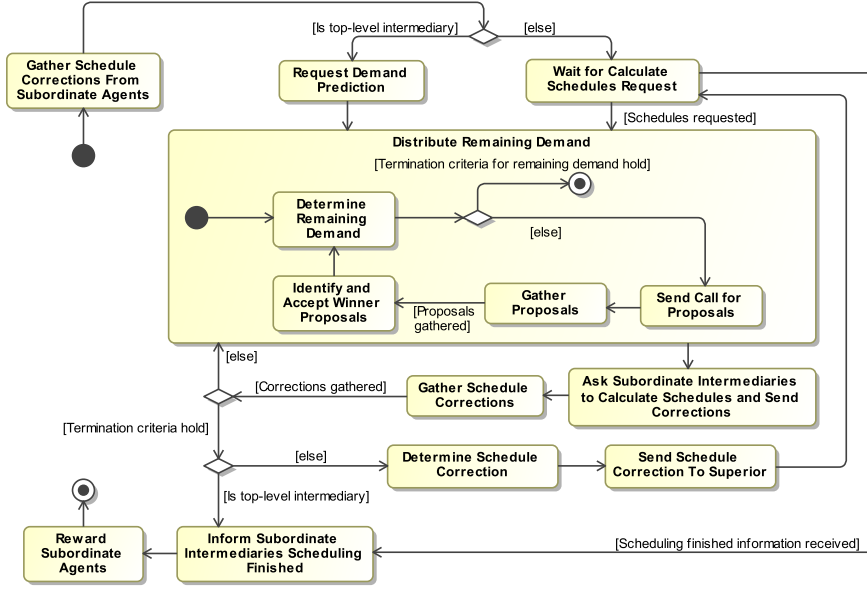


Figure 10.1: TruCAOS's basic procedure.

Distribute Remaining Demand

In all bidding iterations, an intermediary λ has the goal to increase the satisfaction of their fraction of the overall demand stipulated in their scheduled supply S_λ (the schedule S_λ of the top-level intermediary Λ is equivalent to the aggregated demand prediction \hat{D}^W). To this end, λ announces an auction in which its subordinate dispatchable agents $a \in \mathcal{V}_\lambda$ can bid for a part of the *remaining demand* $D_\lambda^{rem}[t] = S_\lambda[t] - \hat{S}_{\mathcal{V}_\lambda}[t]$ that λ has to distribute (note that $D_\lambda^{rem}[t]$ might also be negative in some time steps). Here, $\hat{S}_{\mathcal{V}_\lambda}[t] = \sum_{a \in \mathcal{V}_\lambda} \hat{S}_a[t]$ is the total promised supply of λ 's directly subordinate dispatchable agents \mathcal{V}_λ . The corresponding *call for proposals* (CFP) contains a fraction $g \in [0, 1]$ of D_λ^{rem} and is sent to all $a \in \mathcal{V}_\lambda$.² If $g < 1$, the demand is distributed among the best bidders in the course of multiple bidding iterations, resulting in a fairer allocation of resources, in particular among agents whose proposals are of similar quality. For the sake of simplicity, we deliberately describe the algorithm without making explicit use of this concept so that $CFP = (\langle D_\lambda^{rem}[t_{now} + \Delta\tau], \dots, D_\lambda^{rem}[t_{now} + H] \rangle, \mathcal{W})$ is a 2-tuple consisting of the remaining demand D_λ^{rem} and information about the regarded scheduling window \mathcal{W} .

Each agent $a \in \mathcal{V}_\lambda$ that receives the CFP and wants to sell or buy resources to or from λ responds with a proposal, i.e., a **proposed schedule**, $\hat{S}_a^{pro} = (\langle \hat{S}_a^{pro}[t_{now} + \Delta\tau], \dots, \hat{S}_a^{pro}[t_{now} + H] \rangle, \bar{\kappa}_a^{pro})$ that includes a promised supply $\hat{S}_a^{pro}[t]$ for all $t \in \mathcal{W}$ as well as the *average costs* $\bar{\kappa}_a^{pro}$, i.e., a remuneration, for making a contribution. In the power management case study, $\bar{\kappa}_a^{pro}$ is the unit price of electricity, i.e., the price per kilowatt-hour. For example, a power plant could send a proposal $\hat{S}_a^{pro} = (\langle 20 \text{ MW}, 40 \text{ MW}, 20 \text{ MW} \rangle, 15 \frac{\text{euro cent}}{\text{kWh}})$ for a scheduling window comprising three time steps. While the costs of the proposed supply $\hat{S}_a^{pro}[t]$ in a time step $t \in \mathcal{W}$ are thus given as $\kappa(\hat{S}_a^{pro}, t) = \bar{\kappa}_a^{pro} \cdot |\hat{S}_a^{pro}[t]|$, the total costs of \hat{S}_a^{pro} are defined as the sum $\kappa(\hat{S}_a^{pro}) = \sum_{t \in \mathcal{W}} \kappa(\hat{S}_a^{pro}, t)$ of the costs of the supply over all time steps in \mathcal{W} . Note that a proposer could also offer money to its intermediary (then $\bar{\kappa}_a^{pro} < 0$). For example, this might be the case if it wants to lower its contribution or buy resources.

Having gathered the proposals of all bidders in a set \mathcal{P} , λ completes this bidding iteration by identifying and accepting one or more suitable winner proposals $\mathcal{P}_w \subseteq \mathcal{P}$ as explained in Section 10.2.

² In applications that allow for the exclusion of dispatchable agents from this process, the CFP could only be sent to dispatchable agents whose trust value is above a predefined threshold, thereby forming a so-called *Implicit Trusted Community* [37]. In [10], we showed that this measure can incentivize individually rational agents to behave benevolently.

All other proposals are rejected. In the possibly following bidding iteration, all proposers – including the “winners” – are allowed to send new proposals in order to refine their schedules.

The *most recently* accepted proposal \hat{S}_a^{pro} of a dispatchable agent a defines its **promised** schedule $\hat{S}_a = (\langle \hat{S}_a[t_{now} + \Delta\tau], \dots, \hat{S}_a[t_{now} + H] \rangle, \bar{\kappa}_a)$, which is a contract between a and its intermediary λ for all time steps in the scheduling window \mathcal{W} : a has to comply with its promised schedule in exchange for payment. For this reason, whenever a wants to change its scheduled supply, it has to make a proposal \hat{S}_a^{pro} that, if accepted, replaces the previously accepted proposal \hat{S}_a .

In TruCAOS, a schedule \hat{S}_a does not become invalid if a new schedule creation is started. Instead, it is refined in the course of successive schedule creations if $N > 1$. As a result, intermediaries, acting as auctioneers, as well as proposers can rely on their negotiated contracts. When initializing D_λ^{rem} in the first iteration, an intermediary λ therefore subtracts the most recently accepted proposals of all subordinate dispatchable agents $a \in \mathcal{V}_\lambda$ from the demand it has to distribute according to its schedule S_λ . In case the most recently accepted proposal \hat{S}_a of an agent $a \in \mathcal{V}_\lambda$ was accepted in a previous schedule creation, λ presumes that a makes a neutral contribution of $\hat{S}_a[t] = 0$ for each future time step $t \in \mathcal{W}$ that is not covered by \hat{S}_a . Moreover, in case there is no most recently accepted proposal \hat{S}_a for a , λ assumes a neutral contribution of $\hat{S}_a[t'] = 0$ for all $t' \in \mathcal{W}$ as well as neutral costs of $\bar{\kappa}_a = 0$. Note that these assumptions might result in a schedule that contradicts a 's control model, as is the case if $S_a^{min} > 0$ and a cannot be turned off. In such a case, a has to *correct* this infeasible assignment by means of a so-called *schedule correction* that is guaranteed to be accepted by the intermediary. As explained at the end of this section, intermediaries gather such schedule corrections before they trigger the iterative and incremental distribution the demand.

In TruCAOS, the dispatchable agents create their proposals \hat{S}_a^{pro} by solving a constraint optimization problem that is very similar to the one solved by intermediaries in the regio-central scheduling approach (see Section 2.2): Proposals depend on a 's control model including a cost function, its current state, and the remaining demand specified in the CFP. Differences between the constraint optimization problem presented in Equation (2.1) and those solved by the individual agents can stem from additional individual technical constraints and – because an agent might change its behavior at runtime – variable objectives. However, since each dispatchable agent now only has to determine the scheduled supply for itself, TruCAOS mitigates the scalability issues discussed in Sections 1.3 and 2.2. Instead of $O(2^{|\mathcal{V}_\lambda| \cdot N})$, the complexity of solving the scheduling problem for a subsystem \mathcal{A}_λ with dispatchable agents \mathcal{V}_λ is only $O(j_{total} \cdot (|\mathcal{V}_\lambda| \cdot 2^N + 2^{|\mathcal{V}_\lambda|}))$, with $\mathcal{V}_\lambda = \mathcal{P}$, j_{total} the total number of iterations needed to solve the scheduling problem, and $O(2^{|\mathcal{V}_\lambda|})$ the complexity of selecting winner proposals \mathcal{P}_w from \mathcal{P} . Note that we still benefit from a hierarchical approach due to the complexity of proposal selection.³ Because of the top-down creation of schedules, intermediaries submit bids *before* their subordinates. Therefore, they have to be aware of the capabilities of their collective to avoid infeasible schedules that would have to be corrected. Since intermediaries do not impose schedules on subordinates in TruCAOS, they can use the abstracted control model of their collective to create their own proposals in $O(2^N)$ instead of $O(2^{|\mathcal{V}_\lambda| \cdot N})$ (see Chapter 6).

The iterative process of distributing the remaining demand D_λ^{rem} terminates either if it is sufficiently satisfied (i.e., if its absolute values are below a predefined threshold D_{max}^{rem} so that $\forall t \in \mathcal{W} : |D_\lambda^{rem}[t]| \leq D_{max}^{rem}$), if λ did not receive any proposals (i.e., $\mathcal{P} = \emptyset$), if there is no proposal that increases the demand's satisfaction, or if a maximum number of bidding iterations j_{max} is exceeded. The information about termination originates from the top-level intermediary and propagates downwards in the hierarchy.

Ask Subordinate Intermediaries to Calculate Schedules and Send Corrections

Subsequent to the bidding phase, the intermediary λ asks its directly subordinate intermediaries \mathcal{I}_λ to solve the scheduling problem for their own subsystems. Whenever an intermediary $\lambda' \in \mathcal{I}_\lambda$ receives such a request, it starts the procedure we outlined above, i.e., it distributes its remaining demand in a number of bidding iterations until at least one termination criterion holds.

³Bear in mind that model abstraction requires additional computational effort (see Chapter 6).

As stated above, intermediaries (except for the top-level intermediary) have to allocate resources when taking part in the schedule creation of their superior *before* they know which contributions their subordinate agents actually can or want to make. To obtain a scalable approach, intermediaries have to rely on abstracted control models when creating proposals. However, since abstracted models do not always perfectly reflect the represented collective’s capabilities, abstraction errors might cause situations in which the demand cannot be completely satisfied at the end of the bidding iterations. In such a situation, an intermediary has to inform its superior about the mismatch between the allocated resources of its subordinate agents and its own promised supply, i.e., the final remaining demand. Otherwise, it could not adhere to its promised schedule. This information is sent in the form of a *schedule correction* $\Delta S_\lambda = \langle \Delta S_\lambda[t_{\text{now}} + \Delta\tau], \dots, \Delta S_\lambda[t_{\text{now}} + H] \rangle$ to the superior. The promised \hat{S}_λ and scheduled S_λ contributions are modified accordingly by adding each $\Delta S_\lambda[t]$ to its counterparts $\hat{S}_\lambda[t]$ and $S_\lambda[t]$. Furthermore, the superior adjusts the average costs $\bar{\kappa}_\lambda$ of \hat{S}_λ in such a way that \hat{S}_λ ’s total costs (i.e., the stipulated remuneration) decrease by $\sum_{t \in \mathcal{W}} |\bar{\kappa}_\lambda| \cdot |\Delta S_\lambda[t]|$. Agents are thus incentivized to avoid corrections. Having sent a schedule correction, an intermediary waits for further instructions from its superior.

After collecting the corrections of all subordinate intermediaries, the intermediary adds the corrections to the remaining demand and adjusts the promised schedules as described above. If the remaining demand has changed, is not satisfied, and the maximum number of *correction iterations* k_{max} is not exceeded, the intermediary starts a new bidding phase as explained before in order to reallocate the received corrections. This procedure ensures that corrections made at lower levels do only propagate to the next higher level if other subordinate agents are not able to adjust their supply. This is advantageous as it decreases the probability that, first, the intermediary has to send a correction to its superior (which would reduce its reward) and, second, that more agents are involved in solving the problem. Moreover, due to limiting the locality to one intermediary first, several corrections can be dealt with in parallel before escalating to the next higher level.

If the top-level intermediary observes that at least one termination criterion is met, it informs its subordinate intermediaries that the schedule calculation is finished (see discussion below). If the intermediary is not the top-level intermediary, it waits for further instructions from its superior.

Inform Subordinate Intermediaries about Termination and Reward Agents

The information that the calculation of schedules is finished originates from the top-level intermediary and propagates downwards with respect to the hierarchy as each intermediary forwards the information to its subordinate intermediaries.

Subsequent to the schedule creation, an intermediary distributes a reward r_a to each subordinate dispatchable agent $a \in \mathcal{V}_\lambda$ for its supply in the *previous* time step $t_{\text{now}} - \Delta\tau$ that matches the schedule time pattern. The reward depends on a ’s promised supply $\hat{S}_a[t_{\text{now}} - \Delta\tau]$ and on the average costs $\bar{\kappa}_a$ stipulated in the schedule \hat{S}_a that was valid in $t_{\text{now}} - \Delta\tau$:

$$r_a = \kappa(\hat{S}_a, t_{\text{now}} - \Delta\tau) = \bar{\kappa}_a \cdot \left| \hat{S}_a[t_{\text{now}} - \Delta\tau] \right| \quad (10.1)$$

Note that agents must bear the costs resulting from corrections since these affect their net reward. As agents might not adhere to their promised supply, we redefine the reward in Section 10.4.

Gather Schedule Corrections From Subordinate Agents

As stated before, for each future time step $t \in \mathcal{W}$ that is not covered by a most recently accepted proposal \hat{S}_a , an intermediary assumes that the corresponding agent a does not contribute (i.e., $\hat{S}_a[t] = 0$). However, this contribution might not be feasible for a if it contradicts its control model, e.g., its inertia or minimal or maximal supply. Each agent therefore has the opportunity to adjust its promised \hat{S}_a and scheduled S_a supply by sending a schedule correction ΔS_a to its intermediary *before* the iterative process of distributing the remaining demand is started. These schedule corrections are created in a bottom-up manner, meaning that all leaves in the hierarchy create and send their corrections to their

superior intermediary. An intermediary, in turn, aggregates the corrections of its subordinate agents and forwards this aggregated correction to its own superior. In contrast to schedule corrections that are sent by intermediaries if they cannot completely redistribute their fraction of the overall demand, only those contributions can be adjusted that are not already covered by the most recently accepted proposal \hat{S}_a .

Let us assume, e.g., a most recently accepted proposal $\hat{S}_a = (\langle 30 \text{ MW}, 20 \text{ MW}, 20 \text{ MW} \rangle, 11 \frac{\text{euro cent}}{\text{kWh}})$ of power plant a that was created for the previous scheduling window $\mathcal{W}' = \{10:00, 10:15, 10:30\}$. Given the current time 10:00 a.m. and the current scheduling window $\mathcal{W} = \{10:15, 10:30, 10:45\}$, a 's intermediary assumes a neutral contribution of $\hat{S}_a[10:45] = 0 \text{ MW}$ for 10:45 a.m. Hence, a 's most recently accepted proposal is implicitly updated to $\hat{S}_a = (\langle 20 \text{ MW}, 20 \text{ MW}, 0 \text{ MW} \rangle, 11 \frac{\text{euro cent}}{\text{kWh}})$. If we assume that the power plant has a maximal rate of change of $10 \frac{\text{MW}}{15 \text{ min}}$, decreasing the output from 20 MW to 0 MW within 15 min contradicts its control model so that a schedule correction $\Delta S_a = \langle 0 \text{ MW}, 0 \text{ MW}, 10 \text{ MW} \rangle$ is needed. A schedule correction $\Delta S_a = \langle 0 \text{ MW}, -10 \text{ MW}, 0 \text{ MW} \rangle$ is not allowed because corrections are restricted to the output scheduled for 10:45 a.m.

Again, the promised \hat{S}_a and scheduled S_a contributions are modified according to the correction ΔS_a . The average costs $\bar{\kappa}_a$ stipulated in \hat{S}_a are changed so that \hat{S}_a 's total costs (i.e., the agent's remuneration) increase by $\sum_{t \in \mathcal{W}} \frac{1}{2} \cdot \bar{\kappa}_a^{\min} \cdot \Delta S_a[t]$. Note that this growth is based on $\bar{\kappa}_a^{\min}$, which stands for the *minimum* average costs of the agent's accepted proposals. By multiplying $\bar{\kappa}_a^{\min}$ by the arbitrary decreasing constant $\frac{1}{2}$, the product $\frac{1}{2} \cdot \bar{\kappa}_a^{\min}$ is strictly less than $\bar{\kappa}_a^{\text{avg}}$, which denotes the *mean* of the agent's average costs of accepted proposals.⁴ The agent is thus incentivized to participate in the auctions where it can assess higher costs for its contributions instead of using corrections to increase its reward. If we assume $\bar{\kappa}_a^{\min} = 6 \frac{\text{euro cent}}{\text{kWh}}$ in the above-given example, the power plant's schedule correction $\Delta S_a = \langle 0 \text{ MW}, 0 \text{ MW}, 10 \text{ MW} \rangle$ results in the updated promised schedule $\hat{S}_a = (\langle 20 \text{ MW}, 20 \text{ MW}, 10 \text{ MW} \rangle, 9.4 \frac{\text{euro cent}}{\text{kWh}})$. Note that the average costs decreased from $11 \frac{\text{euro cent}}{\text{kWh}}$ to $9.4 \frac{\text{euro cent}}{\text{kWh}} = \frac{11 \cdot (20+20) + 0.5 \cdot 6 \cdot 10}{20+20+10} \frac{\text{euro cent}}{\text{kWh}}$.

The principle of schedule corrections allows agents to communicate a scheduled supply that has not been accepted in the form of a proposal. Consequently, TruCAOS can also deal with dispatchable agents that *have* to make a contribution regardless of whether their proposals have been accepted or not, such as washing machines and other so-called "deferrable consumers". An in-depth investigation of such types of participants is a part of future work.

10.2 Identification of Winner Proposals

The regarded scheduling problem is a multi-objective optimization problem. While the primary objective is to satisfy the demand as accurately as possible, the secondary objective is to achieve the primary objective at minimal costs (see Equation (1.3a)). TruCAOS solves the scheduling problem by means of a multi-stage decision process that is based on heuristics. It is evident that this procedure does not guarantee to find optimal solutions. But, as stated at the beginning of this chapter, optimal solutions are not required due to the environment's dynamic uncertain nature which requires to revise the schedules over time.

The intermediary determines the set of winner proposals $\mathcal{P}_w \subseteq \mathcal{P}$ on the basis of a three-step process depicted in Figure 10.2. First, invalid proposals are filtered out, then the quality of the remaining proposals is assessed and low-quality proposals are sorted out, and finally the best suited proposals are accepted:

1. *Filter out Invalid Proposals:* Proposals that do not improve the overall satisfaction of the total remaining demand by a minimum value are sorted out.
2. *Filter out Low-Quality Proposals:* The proposals' quality is assessed with regard to their price-performance ratio. Only those proposals whose quality is better than a dynamic threshold pass this filtering stage. If no such proposal exists, the best-rated proposals are passed to the third step in order to allow the algorithm to make any progress.

⁴For the sake of simplicity, we assume here that $\bar{\kappa}_a^{\min}, \bar{\kappa}_a^{\text{avg}} > 0$. The minimum average costs $\bar{\kappa}_a^{\min}$ as well as the mean of the average costs $\bar{\kappa}_a^{\text{avg}}$ are calculated over a fixed number of the last time steps.



Figure 10.2: The steps performed for the activity “Identify and Accept Winner Proposals” in Figure 10.1.

3. *Determine Winner Proposals:* The set of winner proposals \mathcal{P}_w is determined by solving a combinatorial optimization problem in which the gain in satisfaction of the total remaining demand is to be maximized. This objective should be achieved at minimal costs. These objectives as well as their lexicographical order correspond to those of the scheduling problem’s definition. The preceding filtering stages reduce the complexity of the combinatorial optimization problem, which depends on the number of acceptable proposals.

Proposals that do not pass the filtering stages or are not selected as a winner proposal are rejected. Having introduced the basic idea of each of the three steps, we explain them in more detail in the following subsections.

Filter out Invalid Proposals

To satisfy the demand, only those proposals $\hat{S}_a^{pro} \in \mathcal{P}$ that would improve the overall satisfaction of the total remaining demand by a minimum value of $\mathcal{G}_{\min} > 0$ are regarded as valid and pass the filtering stage. To identify which proposals improve the satisfaction, we define $D_\lambda^{pro}(\hat{S}_a^{pro}, t) = D_\lambda^{rem}[t] + \hat{S}_a[t] - \hat{S}_a^{pro}[t]$ as the remaining demand for time step $t \in \mathcal{W}$ that would result if \hat{S}_a^{pro} was accepted and thus replaced \hat{S}_a . The resulting improvement is called the *gain in satisfaction* $\mathcal{G}(\hat{S}_a^{pro})$ and is defined as follows:

$$\mathcal{G}(\hat{S}_a^{pro}) = \sum_{t \in \mathcal{W}} |D_\lambda^{rem}[t]| - |D_\lambda^{pro}(\hat{S}_a^{pro}, t)|$$

As stated above, a proposal \hat{S}_a^{pro} is valid if $\mathcal{G}(\hat{S}_a^{pro}) \geq \mathcal{G}_{\min}$ holds.

Filter out Low-Quality Proposals

In order to keep the costs down, the quality $\mathcal{Q}(\hat{S}_a^{pro})$ of each proposal \hat{S}_a^{pro} is evaluated with respect to its gain in satisfaction as well as the change in costs $\Delta\kappa(\hat{S}_a^{pro})$ that would result in case \hat{S}_a^{pro} was accepted and replaced \hat{S}_a . The quality $\mathcal{Q}(\hat{S}_a^{pro})$ is thus defined as a tuple:

$$\begin{aligned} \mathcal{Q}(\hat{S}_a^{pro}) &= \left(\mathcal{G}(\hat{S}_a^{pro}), \Delta\kappa(\hat{S}_a^{pro}) \right) \\ \text{with } \Delta\kappa(\hat{S}_a^{pro}) &= \sum_{t \in \mathcal{W}} \kappa(\hat{S}_a^{pro}, t) - \kappa(\hat{S}_a, t) \end{aligned}$$

To compare different proposals with regard to the above-mentioned quality criteria, we define a total order on the proposals where we distinguish proposals based on their cost impact. The previous decision stage guarantees that the gain in satisfaction of all proposals is positive in this stage. All proposals \hat{S}_a^{pro} with $\Delta\kappa(\hat{S}_a^{pro}) > 0$ (i.e., the proposals that offer a better satisfaction for higher costs) can therefore be sorted according to their *price-performance ratio* $\Theta(\hat{S}_a^{pro})$, which is defined as the ratio of a proposal’s gain in satisfaction $\mathcal{G}(\hat{S}_a^{pro})$ to its change in costs $\Delta\kappa(\hat{S}_a^{pro})$:

$$\Theta(\hat{S}_a^{pro}) = \frac{\mathcal{G}(\hat{S}_a^{pro})}{\Delta\kappa(\hat{S}_a^{pro})}$$

The greater the price-performance ratio, the higher the quality of the proposal. While this measure allows smaller dispatchable agents with very limited resources to compete with larger dispatchable agents, it might increase the number of bidding iterations needed to satisfy the demand. In case the change

in costs $\Delta\kappa(\hat{S}_a^{pro})$ of a proposal is not positive (i.e., we get a better satisfaction while not facing *any* additional costs but possibly even cost reductions), we do not assess the proposal's quality on the basis of its price-performance-ratio but define that the quality is the higher, the smaller $\Delta\kappa(\hat{S}_a^{pro})$. We further specify that a proposal with $\Delta\kappa(\hat{S}_a^{pro}) \leq 0$ is better than any proposal with positive change in costs.

Together, these properties yield a quality relation $\hat{S}_{a_1}^{pro} \succ \hat{S}_{a_2}^{pro}$ in which a proposal $\hat{S}_{a_1}^{pro}$ is “better” than another proposal $\hat{S}_{a_2}^{pro}$ if and only if the following condition holds:

$$\left(\Delta\kappa(\hat{S}_{a_1}^{pro}) \leq 0 \wedge \Delta\kappa(\hat{S}_{a_1}^{pro}) < \Delta\kappa(\hat{S}_{a_2}^{pro}) \right) \vee \left(\Delta\kappa(\hat{S}_{a_1}^{pro}) > 0 \wedge \Delta\kappa(\hat{S}_{a_2}^{pro}) > 0 \wedge \Theta(\hat{S}_{a_1}^{pro}) > \Theta(\hat{S}_{a_2}^{pro}) \right)$$

Clearly, a *merit order* approach that can only accept the proposals of highest quality leads to low-priced allocations. However, it is computationally inefficient because it needs a large number of bidding iterations to satisfy the demand, which is also accompanied by an increase in the number of messages sent and proposals to generate. For the selection of winner proposals \mathcal{P}_w , we therefore opt for a compromise between the number of bidding iterations and resulting monetary costs. In particular, we enable intermediaries to accept a *combination* of proposals, which allows for taking advantage of synergy effects (these are especially beneficial in the context of multi-objective optimization) and reduces the overhead introduced by additional bidding iterations. To ensure that acceptable proposals and thus the overall result feature a sufficient quality, each intermediary keeps track of the proposals it accepted in a fixed number of the last schedule creations and evaluates the average of each quality criterion separately. Based on this information, only those proposals whose quality is not worse than the average quality of accepted proposals can be accepted in the following decision stage “Determine Winner Proposals”. While this procedure does not yield minimal costs, it ensures that the average quality in terms of costs does not increase. It inherently assumes that there might be less expensive yet suitable proposals in the next bidding iteration. In case no proposal is rated better than the average, only the proposals of highest quality pass the filtering stage to allow the algorithm to make any progress.

Determine Winner Proposals

In case only a single proposal is available in this stage, this proposal is accepted and wins the auction of this iteration. Otherwise, the intermediary determines the set of winner proposals \mathcal{P}_w by solving a multi-objective combinatorial optimization problem, that is, the so-called *combinatorial auction problem*. More precisely, the intermediary chooses \mathcal{P}_w in such a way that there is no other combination of acceptable proposals that yields a greater gain in satisfaction of the total remaining demand. The second and subordinate objective is to achieve the primary goal at a minimal aggregated change in costs (i.e., $\sum_{\hat{S}_a^{pro} \in \mathcal{P}_w} \Delta\kappa(\hat{S}_a^{pro})$ should be as low as possible). These objectives as well as their lexicographical order correspond to the definition of the scheduling problem in Equation (1.2).

Finally, for all $t \in \mathcal{W}$, the total promised supply $\hat{S}_{\lambda}[t]$ is updated according to the sum of the accepted proposed contributions $\sum_{\hat{S}_a^{pro} \in \mathcal{P}_w} \hat{S}_a^{pro}[t]$ of all winner proposals \mathcal{P}_w .

Note – Identifying, Dealing with, and Leaving Local Optima

When solving the combinatorial auction problem, an intermediary might realize that its subsystem is situated in a local optimum in the sense that no dispatchable agent can propose to change its scheduled supply in a way that unilaterally improves the satisfaction of the total remaining demand (and thus satisfies the condition $\mathcal{G}(\hat{S}_a^{pro}) \geq \mathcal{G}_{\min}$ of the first filtering stage). To avoid such a situation, the intermediary would have to know which proposals will actually be made by its subordinate agents *in advance* so that it knew which proposals to accept or reject without getting stuck in a local optimum in subsequent bidding iterations.

Once being trapped in a local optimum, it is very unlikely that the affected subsystem can leave it if the agents propose to change their schedule independently of the decisions of the others. If we wanted to solve the problem in a decentralized manner, the agents would thus have to coordinate their decisions. In our approach, an intermediary meets this challenge by making use of the control models

of its subordinate dispatchable agents (recall that an intermediary uses an abstracted version of these control models to create its own proposals). More precisely, an intermediary can use these control models (1) to identify if its subsystem is in a local optimum and (2) to propose schedule corrections to its subordinate dispatchable agents that actually allow the subsystem to leave the local optimum (evidently, such corrections should be as small as possible).

Basically, the intermediary creates suggested schedule corrections by solving the scheduling problem by means of the regio-central approach (see Section 2.2) with additional constraints that minimize changes. If no suggestions can be generated that increase the demand’s satisfaction, the subsystem does not seem to be able to satisfy the demand the intermediary scheduled when taking part in the schedule creation of its superior. This situation can result from abstraction errors. In such a situation, the intermediary communicates the remaining demand to its superior as described in Section 10.1. The superior, in turn, tries to redistribute the remaining demand to other subordinate dispatchable agents. Otherwise, if the intermediary can identify suitable changes that improve the demand’s satisfaction, it sends them in the form of a schedule correction ΔS_a to the corresponding subordinate agents. Along with such a correction, the intermediary suggests a reward for adjusting the supply. To be more specific, it suggests to change the average costs $\bar{\kappa}_a$ of the corresponding agent’s most recently accepted proposal \hat{S}_a in such a way that the total costs of \hat{S}_a increase by $\sum_{t \in \mathcal{W}} \bar{\kappa}_a^{\text{avg}} \cdot |\Delta S_a[t]|$ (we assume that $\bar{\kappa}_a^{\text{avg}} > 0$). Since the agents’ scheduled contributions have to comply with their control models and we do not assume that intermediaries have perfect knowledge about them (e.g., due to abstraction errors in case of subordinate intermediaries), the agents are allowed to adjust suggested corrections. After the agents sent the adjusted corrections back to the intermediary, it selects and accepts suitable corrections that allow the subsystem to leave the local optimum with a procedure similar to the one described in “Determine Winner Proposals” (see above).

10.3 Trust Metrics for Quantifying and Anticipating Uncertainties in Promised Supply and Predicted Demand

To cope with uncertainties stemming from predicted demand and promised supply, we extend TruCAOS by additional measures based on computational trust. In this section, we devise trust metrics that allow intermediaries to identify and anticipate uncertainties in the form of deviations between predicted and actual demand or promised and actual supply. These metrics are based on the basic trust model we presented in Section 4.2. Based on these metrics, we introduce different measures the intermediaries employ to deal with uncertainties at runtime (see Section 10.4). Some of these measures incentivize dispatchable agents that show avoidable misbehavior to behave benevolently.

A Trust Metric to Quantify and Anticipate Uncertainties in Promised Supply

With respect to uncertain supply, TruCAOS uses trust values to assess the risk that a dispatchable agent violates its contract, i.e., its promised schedule \hat{S}_a . In each time step t that matches the schedule time pattern and for each subordinate dispatchable agent $a \in \mathcal{V}_\lambda$, an intermediary λ gains an experience $E_a[t] = (S_a^{\text{act}}[t], \hat{S}_a[t], t - \Delta\tau, t)$ that captures the deviation $\delta(E_a[t]) = S_a^{\text{act}}[t] - \hat{S}_a[t]$ between a ’s actual $S_a^{\text{act}}[t]$ and promised $\hat{S}_a[t]$ supply at time step t .⁵ An intermediary holds the n_a newest experiences with a dispatchable agent a in a sequence $\mathcal{E}_a = \langle E_a[t_{\text{now}} - (n_a + 1) \cdot \Delta\tau], \dots, E_a[t_{\text{now}}] \rangle$. As is the case with the basic trust model, we only regard the n_a newest experiences because an agent’s behavior may change over time and we assume that more recent behavior can give a better indication of an agent’s future behavior (see Section 4.2).

⁵Note that intermediaries only evaluate the quality of promised contributions made for the next time step that matches the schedule time pattern. Given that schedules are periodically revised, we focus on short-term behavior. Here we assume that schedules are revised after the time span $\Delta\tau$. Intermediaries thus only gain one experience per dispatchable agent and time step.

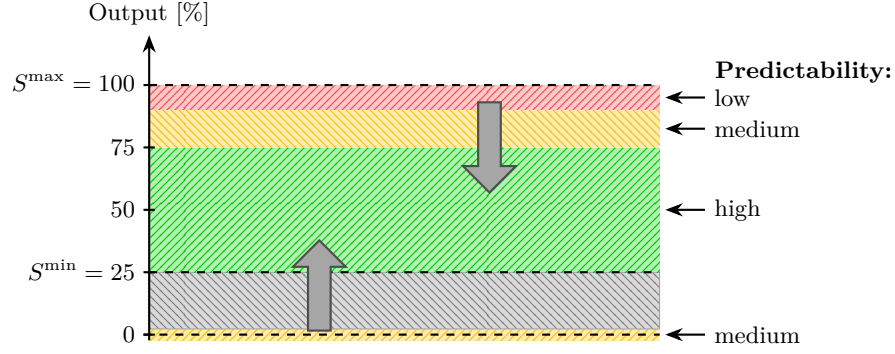


Figure 10.3: A biogas power plant's predictability dependent on its output: We assume that the power plant's gas storage is constantly filled at a rate r . Usually, such a power plant can burn gas at a higher rate than r , which allows the operator to regulate the pressure in the storage by adjusting the power plant's output. If the storage's capacity reaches an upper bound, the power plant's output has to be increased in order to decrease the pressure in its gas storage. Note that this measure might violate the power plant's schedule (i.e., it misbehaves). As a consequence, the power plant features a medium predictability in situations in which it should be turned off. On the other hand, if the power plant runs at (almost) full capacity for a longer period of time, the gas pressure can reach a lower bound. In such a situation, the power plant might violate its schedule since it has to decrease its output. This causes a medium to low predictability at higher output levels. To reduce aleatoric uncertainties originating from this power plant, its scheduled output should be shifted into regions of higher predictability.

To improve the intermediaries' ability to anticipate a dispatchable agent's actual supply, we extend our basic trust model in two ways: First, we introduce a trust context that allows an intermediary to assess a dispatchable agent's trustworthiness – in terms of its mean deviation from promised supply $T_\mu^w(\mathcal{E}_a^c)$ as well as its predictability $T_v^w(\mathcal{E}_a^{\Delta c, c})$ – dependent on a certain stipulated supply c . This is achieved by deriving $T_\mu^w(\mathcal{E}_a^c)$ and $T_v^w(\mathcal{E}_a^{\Delta c, c})$ from the set of experiences $\mathcal{E}_a^c \subseteq \mathcal{E}_a$ that contains the $n'_a \leq n_a$ experiences that are – with regard to their promised supply $\hat{S}_a[t]$ – most similar to c (n'_a being constant). That way, the intermediary can detect situations in which a 's behavior depends on the amount of resources c it professes to provide in the future. For instance, because of technical difficulties, there might be a power plant that makes rather accurate predictions in case its promised supply is rather low, but very inaccurate predictions if its promised supply is high. Figure 10.3 depicts an example of a biogas power plant that features an output-specific predictability.

Second, because of the dispatchable agents' possibly contribution-specific and dynamic behavior, we weight experiences when calculating the trust values. More precisely, we weight each experience $E_a[t] \in \mathcal{E}_a^c$ according to the dissimilarity $\Delta_{\text{dis}}(E_a[t]) = |c - \hat{S}_a[t]|$ between the given supply c and $E_a[t]$'s promised supply $\hat{S}_a[t]$ on the one hand, and according to its age $\Delta_{\text{age}}(E_a[t]) = t_{\text{now}} - t$ on the other hand. The less similar $E_a[t]$'s promised supply is to c and the older $E_a[t]$, the higher the values for $\Delta_{\text{age/dis}}$ and, consequently, the lower should be its weight.⁶

The function $w_{\text{age/dis}}(E_a[t], \mathcal{E}_a^c) \in [0, 1]$ summarizes these properties in a more formal way. It determines the weight of an experience $E_a[t]$ either with regard to its age or dissimilarity to c (in this context, $\Delta_{\text{age/dis}}^{\text{max}}(\mathcal{E}_a^c)$ yields the maximum $\Delta_{\text{age}}(E_a[t])$ or $\Delta_{\text{dis}}(E_a[t])$ of all experiences in \mathcal{E}_a^c ; smallest values are provided by $\Delta_{\text{age/dis}}^{\text{min}}(\mathcal{E}_a^c)$):

$$w_{\text{age/dis}}(E_a[t], \mathcal{E}_a^c) = \begin{cases} \frac{\Delta_{\text{age/dis}}^{\text{max}}(\mathcal{E}_a^c) - \Delta_{\text{age/dis}}(E_a[t])}{\Delta_{\text{age/dis}}^{\text{max}}(\mathcal{E}_a^c) - \Delta_{\text{age/dis}}^{\text{min}}(\mathcal{E}_a^c)} & \text{if } \Delta_{\text{age/dis}}^{\text{max}}(\mathcal{E}_a^c) \neq \Delta_{\text{age/dis}}^{\text{min}}(\mathcal{E}_a^c) \\ 1 & \text{else} \end{cases}$$

⁶Note that we write “age/dis” to either refer to “age” or “dissimilarity”.

The actual weight $w(E_a[t], \mathcal{E}_a^c) = \frac{w_{\text{age}}(E_a[t], \mathcal{E}_a^c) + w_{\text{dis}}(E_a[t], \mathcal{E}_a^c)}{2}$ of an experience is the mean of its age and dissimilarity weights $w_{\text{age}}(E_a[t], \mathcal{E}_a^c)$ and $w_{\text{dis}}(E_a[t], \mathcal{E}_a^c)$.

Because of a dispatchable agent's possibly contribution-specific behavior, we use the concept of *impartiality* with regard to its behavior in a certain contribution range Δc : If there is no experience $E_a[t]$ with $\Delta_{\text{dis}}(E_a[t]) \leq \Delta c$, the intermediary adds an artificial experience $E_a[t_{\text{now}}] = (0, 0, t_{\text{now}}, t_{\text{now}})$ of assumed benevolent behavior to \mathcal{E}_a^c . When calculating the trust value, this experience has the greatest influence on the result. This principle of relying on artificial experiences in certain situations is similar to the well-known concept of *initial trust* [144] (see Section 4.1).

Based on the experiences \mathcal{E}_a^c , the trust value $T_\mu^w(\mathcal{E}_a^c)$ represents the expectation that a dispatchable agent a makes a contribution c . It is defined as the weighted mean of measured differences between actual and promised supply that have been recorded in the experiences $E_a[t]$:

$$T_\mu^w(\mathcal{E}_a^c) = \begin{cases} \frac{1}{\sum_{E_a[t] \in \mathcal{E}_a^c} w(E_a[t])} \cdot \sum_{E_a[t] \in \mathcal{E}_a^c} \left(w(E_a[t]) \cdot \frac{\delta(E_a[t])}{\delta_{\mathcal{V}_\lambda}^{\max}} \right) & \text{if } \delta_{\mathcal{V}_\lambda}^{\max} > 0 \\ 0 & \text{else} \end{cases}$$

To assess the behavior of a in relation to the behavior of its siblings $\mathcal{V}_\lambda \setminus \{a\}$, an intermediary λ uses the maximum absolute deviation $\delta_{\mathcal{V}_\lambda}^{\max} = \max \{ |\delta(E_{a'}[t])| \mid E_{a'}[t] \in \mathcal{E}_{a'} \wedge a' \in \mathcal{V}_\lambda \}$ it observed as normalization factor. As is the case with our basic trust metric (see Equation (4.1)), $T_\mu^w(\mathcal{E}_a^c)$ is from the interval $[-1, 1]$ and represents a 's *expected deviation* $\mathbb{E}(\delta_a^c) = T_\mu^w(\mathcal{E}_a^c) \cdot \delta_{\mathcal{V}_\lambda}^{\max}$ from a stipulated supply $c = \hat{S}_a^{(pro)}[t]$ (we use " $\hat{S}_a^{(pro)}[t]$ " to refer to a promised $\hat{S}_a[t]$ or a proposed supply $\hat{S}_a^{pro}[t]$). The greater $|T_\mu^w(\mathcal{E}_a^c)|$, the less trustworthy a . If $T_\mu^w(\mathcal{E}_a^c) < 0$, λ expects that a 's actual supply $S_a^{act}[t]$ will be lower than $\hat{S}_a^{(pro)}[t]$. If $T_\mu^w(\mathcal{E}_a^c) > 0$, λ expects that $S_a^{act}[t]$ is greater than $\hat{S}_a^{(pro)}[t]$. Otherwise if $T_\mu^w(\mathcal{E}_a^c) = 0$, λ expects a to comply with its stipulated supply (i.e., $S_a^{act}[t] = \hat{S}_a^{(pro)}[t]$).

As introduced in our basic trust model, we use the concept of *predictability* to assess the risk that an agent does not make a contribution as stipulated or expected. In line with the trust value $T_\mu^w(\mathcal{E}_a^c)$ – the *weighted* mean of observed deviations –, we define the predictability $T_v^w(\mathcal{E}_a^{\Delta c, c}) \in [0, 1]$ of an agent a in the context of a supply c as the corresponding *weighted* standard deviation of the experiences $\mathcal{E}_a^{\Delta c, c} \subseteq \mathcal{E}_a^c$. The set $\mathcal{E}_a^{\Delta c, c}$ only contains those experiences $E_a[t] \in \mathcal{E}_a^c$ whose dissimilarity $\Delta_{\text{dis}}(E_a[t])$ is in a certain range $[0, \Delta c]$ (this set is non-empty since it contains at least the artificial experience). That way, we take account of the characteristic that an agent's behavior can depend on its stipulated supply. If we did not focus on those experiences that satisfy $\Delta_{\text{dis}}(E_a[t]) < [0, \Delta c]$, the standard deviation of experiences in \mathcal{E}_a^c could be very high although the agent's behavior is quite predictable in different contribution ranges. Again, expected variations are defined as $v(\delta_a^{\Delta c, c}) = T_v^w(\mathcal{E}_a^{\Delta c, c}) \cdot \delta_{\mathcal{V}_\lambda}^{\max}$.

A Trust Metric to Quantify and Anticipate Uncertainties in Predicted Demand

Please note that, in this chapter, only the top-level intermediary Λ assesses the quality of aggregated demand predictions provided by non-dispatchable agents. For this purpose, it uses the trust metric outlined above. However, since we assume that predictions become more accurate as the point in time they are made for approaches, we integrate the prediction horizon Δh as trust context – analogously to our basic trust model in Section 4.2.

As a consequence, the intermediary evaluates the expected deviation $\mathbb{E}(\delta_D^{\Delta h, d}) = T_\mu^w(\mathcal{E}_D^{\Delta h, d}) \cdot \delta_D^{\max}$ from a demand prediction $d = \hat{D}^\mathcal{W}[t]$ (with $t \in \mathcal{W}$) in the context of the temporal distance $\Delta h = t - t_{\text{now}}$. The trust value $T_\mu^w(\mathcal{E}_D^{\Delta h, d}) \in [-1, 1]$ is derived from the experiences $\mathcal{E}_D^{\Delta h, d} \subseteq \mathcal{E}_D^{\Delta h}$. While $\mathcal{E}_D^{\Delta h}$ contains the n_D newest experiences, $\mathcal{E}_D^{\Delta h, d}$ comprises the $n'_D \leq n_D$ experiences that are – in terms of the predicted demand – most similar to d (n'_D being constant). The maximum absolute deviation from a demand prediction, $\delta_D^{\max} \geq 0$, serves as normalization factor.

10.4 Trust-Based Measures to Deal with Uncertain Supply and Demand

In TruCAOS, intermediaries use trust values in various ways to identify and deal with uncertainties originating from both non-dispatchable as well as dispatchable agents. On the one hand, TruCAOS's trust-based measures allow the agents to cope with unavoidable misbehavior, such as inaccurate consumption or production predictions due to outdated standard load profiles or inaccurate sensors. On the other hand, TruCAOS imposes indirect, trust-based sanctions as well as direct sanctions in the form of punitive fines in order to lower the utility of (uncooperative) dispatchable agents that show avoidable misbehavior (i.e., deviation that could have been prevented). In case of avoidable misbehavior, these measures incentivize benevolent behavior (e.g., the owner of a biogas plant is incentivized to ensure that the promised output can actually be provided), which corresponds to the idea of *mechanism design* [58] (we discuss the similarities and differences to mechanism design in Chapter 13). We therefore assume that the agents know how misbehavior is sanctioned and that the opportunity cost the sanctions incur imply that it is in their best interest to behave benevolently – i.e., we assume them to be *individually rational*. Under the assumption that the superior of an intermediary λ employs the trust-based measures we describe in the following, it is also in λ 's best interest to maintain a good trust value itself. Since an intermediary's supply is the aggregate of the supply of its subordinate dispatchable agents, it should accept proposals in such a way that the subordinate dispatchable agents' promised contributions are as similar as possible to their actual contributions. The following enumeration summarizes the trust-based measures by which we enhance TruCAOS:

1. *Rewarding Agents:* To incentivize agents to comply with their promised supply, we adjust the reward function defined in Equation (10.1). Intermediaries distribute rewards according to the agents' actual instead of their promised supply. Deviations from promised supply are penalized by lowering the corresponding agent's reward.
2. *Expected Demand and Supply:* Intermediaries use trust values to form expectations of the remaining demand and scheduled supply instead of relying on predictions and promises. As our evaluation shows, scheduling on the basis of expectations reduces the probability of deviations between actual demand and supply because our trust model allows intermediaries to lower epistemic uncertainties (see Section 10.5).
3. *Redefining Valid Proposals:* A proposal is now regarded as valid if the *expected* gain in satisfaction is improved. This expectation is also derived from the information provided by trust values.
4. *Expected Quality and Price-Performance Ratio:* A proposal's quality is now assessed by using an *expectation* of the change in costs on the basis of the idea of the revised reward function. Furthermore, we make use of the concept of predictability. Recall that the predictability represents the risk that the goal of balancing supply and demand is not achieved because the agent does not provide resources as stipulated or expected. The predictability thus quantifies aleatoric uncertainties. Because of an agent's possibly contribution-specific behavior, its predictability might change as a result of accepting its proposal. We therefore map the change in predictability to costs that are incorporated into a proposal's quality assessment. As our evaluation confirms, this measure allows us to efficiently put the agents into states of less aleatoric uncertainties (see Figure 10.3 for an illustration). As before, only those proposals can be accepted that feature a sufficient quality.

Note that only the first measure exclusively addresses avoidable misbehavior. All other measures aim at enabling the system to deal with uncertainties resulting from both avoidable and, in particular, unavoidable misbehavior. We outline these trust-based measures in more detail in the following subsections.

Rewarding Agents

First, as the actual supply $S_a^{act}[t]$ of a dispatchable agent a in a time step t might differ from its promised supply $\hat{S}_a[t]$, we redefine the reward r_a the agent a receives as follows:

$$r_a = \kappa(\hat{S}_a, t_{\text{now}} - \Delta\tau) - \left(\max \{ |\bar{\kappa}_a|, |\bar{\kappa}_{\mathcal{V}_\lambda}^{\text{avg}}| \} \cdot \left| \hat{S}_a[t_{\text{now}} - \Delta\tau] - S_a^{act}[t_{\text{now}} - \Delta\tau] \right| \right) \quad (10.2)$$

In contrast to the definition of the reward function in Equation (10.1), we now charge the agent for the deviation $\left| \hat{S}_a[t_{\text{now}} - \Delta\tau] - S_a^{act}[t_{\text{now}} - \Delta\tau] \right|$ from its promised supply. The punishment is based on the maximum of the absolute value of the stipulated average costs $\bar{\kappa}_a$ in the corresponding schedule \hat{S}_a and the absolute value of the subsystem's mean average costs of accepted proposals $\bar{\kappa}_{\mathcal{V}_\lambda}^{\text{avg}}$. The latter is calculated over a fixed number of the last time steps. Using absolute values ensures that the agent is punished for deviations, regardless of whether its actual supply is higher or lower than its promised supply, thereby incentivizing benevolent behavior. Dispatchable agents that abide by their promises obtain the same reward as before (cf. Equation (10.1)).

Please note that schedule corrections – which an intermediary λ sends to its superior if it cannot completely satisfy its fraction of the overall demand – also lower λ 's reward (see Section 10.1). However, λ 's superior still has the opportunity to compensate for the schedule correction *before* the contribution actually has to be made. The reduction of λ 's reward caused by such schedule corrections is therefore not greater but usually lower than the punishment for a non-communicated deviation from a promised supply.

Expected Demand and Supply

In this version of TruCAOS, the top-level intermediary Λ makes use of the trust values assessing the quality of demand predictions to determine expectations of the future demand (see Section 10.3). The expected demand $\bar{D}^\mathcal{W}[t] = \hat{D}^\mathcal{W}[t] + \mathbb{E}(\delta_D^{\Delta h, d})$ (with $\Delta h = t - t_{\text{now}}$) in a time step $t \in \mathcal{W}$ is defined on the basis of the demand prediction $d = \hat{D}^\mathcal{W}[t]$ and the expected deviation $\mathbb{E}(\delta_D^{\Delta h, d})$ that results from the corresponding trust value $T_\mu^w(\mathcal{E}_D^{\Delta h, d})$. This corresponds to the definition of the expected demand in our basic trust model in Section 4.2.

Analogously, every intermediary λ forms expectations of future contributions. The expected supply $\bar{S}_a^{(pro)}[t] = \hat{S}_a^{(pro)}[t] + \mathbb{E}(\delta_a^c)$ of a dispatchable agent a is defined on the basis of its stipulated supply $c = \hat{S}_a^{(pro)}[t]$ and the expected deviation $\mathbb{E}(\delta_a^c)$ that originates from the corresponding trust value $T_\mu^w(\mathcal{E}_a^c)$ (see Section 10.3). Similarly to \hat{S}_a and \hat{S}_a^{pro} , we define the **expected schedule** $\bar{S}_a = (\langle \bar{S}_a[t_{\text{now}} + \Delta\tau], \dots, \bar{S}_a[t_{\text{now}} + H] \rangle, \bar{\kappa}_a)$ as well as the expected schedule $\bar{S}_a^{pro} = (\langle \bar{S}_a^{pro}[t_{\text{now}} + \Delta\tau], \dots, \bar{S}_a^{pro}[t_{\text{now}} + H] \rangle, \bar{\kappa}_a^{(pro)})$ that would result *in case a proposal \hat{S}_a^{pro} was accepted*. Note that the average costs $\bar{\kappa}_a^{(pro)}$ are the same for $\bar{S}_a^{(pro)}$ and $\hat{S}_a^{(pro)}$. Based on these expectations, the call for proposals now contains the *expected* remaining demand $D_\lambda^{rem}[t] = S_\lambda[t] - \bar{S}_{\mathcal{V}_\lambda}[t]$.

Redefining Valid Proposals

The remaining demand $D_\lambda^{pro}(\hat{S}_a^{pro}, t) = D_\lambda^{rem}[t] + \bar{S}_a[t] - \bar{S}_a^{pro}[t]$ for a time step $t \in \mathcal{W}$ that would result if \hat{S}_a^{pro} was accepted (see Section 10.2) is now based on the expected change in contribution $\bar{S}_a[t] - \bar{S}_a^{pro}[t]$, which results from an agent's trust value $T_\mu^w(\mathcal{E}_a^c)$. The gain in satisfaction $\mathcal{G}(\hat{S}_a^{pro})$ is thus an expectation. Because of contribution-specific uncertainties associated with a dispatchable agent a , note that $\bar{S}_a^{pro}[t]$ might be smaller than $\bar{S}_a[t]$, although $\hat{S}_a^{pro}[t]$ is greater than $\hat{S}_a[t]$. For the same reason, regarding two proposals $\hat{S}_{a_1}^{pro}$ and $\hat{S}_{a_2}^{pro}$ of two agents a_1 and a_2 , the expected gain in satisfaction $\mathcal{G}(\hat{S}_{a_1}^{pro})$ might be higher than $\mathcal{G}(\hat{S}_{a_2}^{pro})$, although the overall supply predicted in $\hat{S}_{a_1}^{pro}$ is smaller than in $\hat{S}_{a_2}^{pro}$.

Expected Quality and Price-Performance Ratio

As defined in Section 10.2, a proposal's quality $\mathcal{Q}(\hat{S}_a^{pro})$ and price-performance ratio $\Theta(\hat{S}_a^{pro})$ are based on the change in costs $\Delta\kappa(\bar{S}_a^{pro})$. Here, we redefine the change in costs to use expectations. More precisely, we seize the idea of the updated reward function (see Equation (10.2)) to determine the expected change in costs $\Delta\kappa(\bar{S}_a^{pro})$. Instead of actual contributions and actual deviations from promised supply (both values are obviously not known in advance), the intermediary uses expected contributions $\bar{S}_a^{(pro)}[t]$ as well as expected deviations $\mathbb{E}(\delta_a^c)$ to evaluate $\Delta\kappa(\bar{S}_a^{pro})$:

$$\begin{aligned}\Delta\kappa(\bar{S}_a^{pro}) &= \sum_{t \in \mathcal{W}} \mathbb{E}(\bar{\kappa}(\bar{S}_a^{pro}, t)) - \mathbb{E}(\bar{\kappa}(\bar{S}_a, t)) \\ \text{with } \mathbb{E}(\bar{\kappa}(\bar{S}_a^{(pro)}, t)) &= \kappa(\bar{S}_a^{(pro)}, t) - (\max\{|\bar{\kappa}_a|, |\bar{\kappa}_{\mathcal{V}_\lambda}^{avg}|\} \cdot |\mathbb{E}(\delta_a^c)|) \\ \text{and } c &= \bar{S}_a^{(pro)}[t]\end{aligned}$$

Apart from the change in costs, a proposal's quality $\mathcal{Q}(\hat{S}_a^{pro})$ and price-performance ratio $\Theta(\hat{S}_a^{pro})$ also rely on the corresponding agent's trustworthiness as the gain in satisfaction $\mathcal{G}(\hat{S}_a^{pro})$ is based on the trust value $T_\mu^w(\mathcal{E}_a^c)$. However, as discussed in Section 4.2, positive and negative deviations can cancel each other out, which might result in a high trust value, quality, and price-performance ratio although the agent's compliance with its proposed supply is hard to predict. To master such situations, we incorporate an agent's predictability $T_v^w(\mathcal{E}_a^{\Delta c, c})$ when assessing the quality of its proposals. Since we want to effectively reduce the presence of aleatoric uncertainties, intermediaries evaluate the *change in predictability* $\Delta_v(\bar{S}_a^{pro})$ when deciding whether a proposal \hat{S}_a^{pro} should be accepted and replace \hat{S}_a :

$$\Delta_v(\bar{S}_a^{pro}) = \sum_{t \in \mathcal{W}} v(\delta_a^{\Delta c, c_1}) - v(\delta_a^{\Delta c, c_2}), \text{ with } c_1 = \bar{S}_a^{pro}[t] \text{ and } c_2 = \bar{S}_a[t]$$

If $\Delta_v(\bar{S}_a^{pro}) < 0$, the intermediary assumes that the predictability increases and the risk of imbalances between demand and supply decreases. By multiplying the change in predictability $\Delta_v(\bar{S}_a^{pro})$ by a factor κ_v , we map the value to costs (e.g., costs that would occur to compensate for a deviation from an expected supply at runtime, such as reserves activated by primary control in power systems). This mapping allows us to incorporate the change in predictability into the intermediary's decisions by adding the costs $\Delta_v(\bar{S}_a^{pro}) \cdot \kappa_v$ to the change in costs $\Delta\kappa(\bar{S}_a^{pro})$. The result is the so-called *risk-based change in costs* $\Delta\kappa_{risk}(\bar{S}_a^{pro})$ that would result in case \hat{S}_a^{pro} was accepted and replaced \hat{S}_a :

$$\Delta\kappa_{risk}(\bar{S}_a^{pro}) = \Delta\kappa(\bar{S}_a^{pro}) + \Delta_v(\bar{S}_a^{pro}) \cdot \kappa_v$$

The greater the factor κ_v , the more risk-averse an intermediary's decisions. Certainly, intermediaries have to come to a compromise between costs and risk. When evaluating the quality $\mathcal{Q}(\hat{S}_a^{pro})$ of a proposal \hat{S}_a^{pro} or its price-performance ratio $\Theta(\hat{S}_a^{pro})$, an intermediary now uses $\Delta\kappa_{risk}(\bar{S}_a^{pro})$ instead of $\Delta\kappa(\bar{S}_a^{pro})$.

Due to these influences of an agent's trustworthiness on the quality and price-performance ratio of its proposals, trustworthy agents can charge higher prices for a contribution than untrustworthy agents. This incentivizes benevolent behavior. In economics, such financial benefits are known as *price premiums* and *price discounts* [25]. This illustrates that trust in the sense of benevolent behavior yields and, at the same time, embodies a form of *social capital* [163].

To summarize, all these measures are aimed at reducing epistemic as well as the presence of aleatoric uncertainties. The latter is achieved on the basis of the concept of predictability, which allows intermediaries to anticipate aleatoric uncertainties with regard to the dispatchable agents' states. In turn, this information is used to actively put the agents into states of less aleatoric uncertainties as illustrated in Figure 10.3.

10.5 Evaluation in the Decentralized Power Management Case Study

We evaluate TruCAOS in the context of our power management case study in two different settings. In the first setting, we primarily investigate in which way TruCAOS can deal with uncertain demand and

supply. In the second setting, we analyze TruCAOS’s runtime performance and the quality of created schedules in a hierarchy of AVPPs. We compare the results to those achieved with the regio-central approach *RegioC* sketched in Section 2.2. For each evaluation scenario regarded in these two settings, we performed 200 simulation runs. Where not stated otherwise, presented data are average values. Before we discuss our results, we introduce our general test bed.

General Test Bed

We perform the evaluation in our simulation environment for autonomous power systems that contains dispatchable and non-dispatchable power plants of different types (hydro, biofuel, and gas power plants as well as solar plants and wind generators). Each power plant is modeled as an individual agent. The consumers are represented by a single agent. The demand, i.e., the residual load, originates from weather-dependent power plants and the consumers. We use real world data for the capabilities of physical power plants⁷ (such as production boundaries), the load curves⁸, and the simulated weather conditions⁹ influencing the output of weather-dependent power plants. Each dispatchable power plant’s inertia is defined within typical boundaries. The production costs, i.e., the average costs of providing a contribution, range from $6.50 \frac{\text{euro cent}}{\text{kWh}}$ to $17.50 \frac{\text{euro cent}}{\text{kWh}}$.

The evaluation is implemented in a sequential, round-based execution model in which each round corresponds to a specific time step $t \in \mathcal{T}$. Dispatchable power plants have to satisfy a prescribed residual load over an evaluation-scenario-specific period of $|\mathcal{T}|$ time steps, each representing a time span of 15 min. In each time step, AVPPs create schedules for the next $H = 1$ hour on the basis of the predicted residual load retrieved from the consumers and the weather-dependent power plants. We also use a resolution of 15 min for the schedules so that each of them comprises $N = 4$ time steps. In case of TruCAOS, AVPPs consider the remaining residual load satisfied in case its absolute values are below $D_{\max}^{\text{rem}} = 1 \text{ kW}$. As for the incremental distribution of the remaining residual load D_{λ}^{rem} , AVPPs use a fraction $g = 0.5$ of D_{λ}^{rem} if $D_{\lambda}^{\text{rem}} \cdot g > 40 \text{ kW}$, and $g = 1$ otherwise.

The optimization problems that have to be solved to generate proposals, determine winner proposals, and leave local optima in TruCAOS, as well as those of the regio-central approach *RegioC* are formulated as mixed integer linear programs and solved by the standard mathematical programming software IBM ILOG CPLEX¹⁰. As the utmost goal is to satisfy the residual load, followed by the goal to provide energy at a low price, we use $\alpha_{\Delta} = 100.0 \frac{\text{euro cent}}{\text{kWh}}$ and $\alpha_{\Gamma} = 1.0$ to weight the total residual load violation Δ and the total costs Γ accordingly (regarding Equation (1.3a)), note that Δ is given in kWh and Γ in euro cent). It is thus more expensive to not produce a requested amount of energy than to produce it. Where not stated otherwise, dispatchable power plants generate proposals that minimize the remaining residual load specified in the AVPPs’ call for proposals in order to be competitive. Physical power plants demand remunerations proportionally to their output and their individual unit price of electricity. Each AVPP uses its subsystem’s mean average costs of accepted proposals $\bar{\kappa}_{\mathcal{V}_{\lambda}}^{\text{avg}}$ (see Equation (10.2)) – calculated over the last 5 time steps – as the average costs $\bar{\kappa}_a^{\text{pro}}$ of its proposals. In line with the definition of the maximum sequential scheduling time, we assume that an intermediary’s subordinate dispatchable power plants create their proposed schedules in parallel.

Dealing with Uncertain Demand and Supply

We evaluate TruCAOS’s performance in uncertain environments by means of a system that contains a single AVPP. For comparison, we regard two different evaluation scenarios. In the first scenario “E-nt”, TruCAOS does not make any use of trust values. We change this in the second scenario “E-t” in which we enable TruCAOS’s trust-based measures ($n_a = 100$, $n'_a = 5$, $n_D = 50$, and $n'_D = 5$ turned out to be beneficial parameters for our trust metrics in a preceding parameter search).

⁷Energymap (Bavaria), 2012: <http://www.energymap.info>, retrieved in 2012.

⁸LEW, 2012: <http://www.lew-verteilnetz.de>, retrieved in 2012.

⁹LfL (Bavaria), 2010: <http://www.lfl.bayern.de/agm/>, retrieved in 2012.

¹⁰IBM ILOG CPLEX Optimizer, Version 12.4, 2011: <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, retrieved on March 1, 2016.

Test Bed

The single AVPP controls 20 dispatchable power plants (hydro, biofuel, and gas power plants) that have different capabilities, e.g., in terms of production boundaries. In each simulation run, these power plants have to satisfy a predefined actual residual load $D[t] \in [1342 \text{ kW}, 2157 \text{ kW}]$, i.e., a demand, over a period of four days, which corresponds to 384 time steps (i.e., $\mathcal{T} = \{1, \dots, 384\}$).

Each time the AVPP creates schedules, the consumers and the weather-dependent power plants create a residual load prediction for the following $N = 4$ time steps by adding a random prediction error – generated using a Gaussian distribution – to the actual residual load. In the course of the four simulated days, the residual load shows two different behaviors rl_1 and rl_2 . For rl_1 , we use a mean prediction error of $\mu_{rl_1} = 50 \text{ kW}$ and a standard deviation of $\sigma_{rl_1} = 5 \text{ kW}$. The residual load behaves according to rl_1 in time steps $t \in \{1, \dots, 48\} \cup \{97, \dots, 144\}$. In time steps $t \in \{49, \dots, 96\} \cup \{145, \dots, 384\}$, we use rl_2 with $\mu_{rl_2} = -100 \text{ kW}$ and $\sigma_{rl_2} = 10 \text{ kW}$. The mean prediction error increases with the prediction horizon by 10 kW for rl_1 and $|-20 \text{ kW}|$ for rl_2 .

Except for a “misbehaving dispatchable power plant” (MPP), which features a maximum output of $S_{\text{MPP}}^{\text{max}} = 450 \text{ kW}$ (this is about a fifth of the maximum residual load in this setting), all dispatchable power plants behave benevolently, meaning that they comply with their promised supply. The MPP’s promised outputs \hat{S}_{MPP} deviate from its actual and scheduled output (here $S_{\text{MPP}}^{\text{act}} = S_{\text{MPP}}$) according to two different predefined behaviors mpp_1 and mpp_2 . Regarding mpp_1 / mpp_2 , the MPP adds a random deviation that is, on average, 30 % / 10 % higher than promised if $\hat{S}_{\text{MPP}}[t] = S_{\text{MPP}}^{\text{min}} = 0 \text{ kW}$, and, on average, 30 % / 10 % lower than promised if $\hat{S}_{\text{MPP}}[t] = S_{\text{MPP}}^{\text{max}} = 450 \text{ kW}$. Again, random numbers are generated using a Gaussian distribution with a standard deviation of $\sigma_{mpp_1} = 13.5 \text{ kW} / \sigma_{mpp_2} = 4.5 \text{ kW}$. Between $S_{\text{MPP}}^{\text{min}}$ and $S_{\text{MPP}}^{\text{max}}$, the deviation changes linearly with $\hat{S}_{\text{MPP}}[t]$ such that the power plant does not deviate from its promises at its sweet spot $\hat{S}_{\text{MPP}}[t] = 225 \text{ kW}$. The MPP shows behavior mpp_1 in time steps $t \in \{1, \dots, 72\}$ and behavior mpp_2 in time steps $t \in \{73, \dots, 384\}$. To estimate the MPP’s predictability, we use $\Delta c = 112.5 \text{ kW}$ (i.e., 25 % of $S_{\text{MPP}}^{\text{max}}$) as contribution range in E-t. To investigate in which way the MPP influences the creation of schedules, the AVPP contains a dispatchable power plant, called the MPP’s “twin”, that is – apart from the fact that it behaves benevolently – identical to the MPP. Because the twin does not deviate from its promises, its behavior is very predictable. The concept of the MPP’s twin allows us to examine TruCAOS’s trust-based decisions independently of the other power plants’ physical capabilities.

Results

In the evaluation scenario E-nt, the AVPP relies on the predicted residual load \hat{D} when creating schedules. As depicted in Figure 10.4, the mean absolute deviation $\Delta(D, \hat{D})$ between the actual D and the predicted \hat{D} residual load varies with the residual load’s behavior rl_1 and rl_2 (the average of $\Delta(D, \hat{D})$ is 60.87 kW with a standard deviation of $\sigma = 15.56 \text{ kW}$). Regarding E-t, the AVPP assesses the quality of residual load predictions by means of the presented trust metric and solves the scheduling problem on the basis of the expected residual load \bar{D} . The small mean absolute deviation $\Delta(D, \bar{D})$ between D and \bar{D} indicates that the AVPP is able to anticipate systematic prediction errors, i.e., epistemic uncertainties, very accurately (the average of $\Delta(D, \bar{D})$ is 11.38 kW , $\sigma = 11.66 \text{ kW}$). The expected residual load \bar{D} is thus 81.30 % more accurate than its prediction \hat{D} .

The rather high values of $\Delta(D, \bar{D})$ around the time steps 48, 96, and 144 coincide with the change in the residual load’s behavior. While $\Delta(D, \bar{D})$ sometimes exceeds $\Delta(D, \hat{D})$ at these time steps, the short time frames in which these deviations are present indicate that the trust values quickly adapt to behavioral changes. This can be attributed to our trust metric that weights experiences according to their age. Further, only the last $n_D = 50$ experiences are used for the calculation of the trust values.

With regard to E-nt, the MPP’s promised supply \hat{S}_{MPP} correlates with the predicted residual load \hat{D} , which is mirrored in the shapes of the two corresponding curves (see Figures 10.4 and 10.5). In time step 72, the change in the MPP’s behavior from mpp_1 to mpp_2 is clearly visible. Figure 10.5 illustrates that the mean absolute deviation $\Delta(S_{\text{MPP}}, \hat{S}_{\text{MPP}})$ between the MPP’s actual S_{MPP} and promised \hat{S}_{MPP}

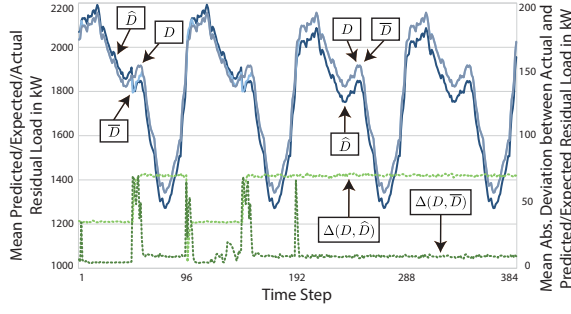


Figure 10.4: The mean predicted \hat{D} , expected \bar{D} , and actual D residual load. The mean absolute deviation $\Delta(D, \hat{D})$ between D and \hat{D} reflects the predefined randomly generated prediction error. As for E-t, the mean absolute deviation $\Delta(D, \bar{D})$ between D and \bar{D} mirrors the accuracy of \bar{D} , which is provided by the presented trust metrics.

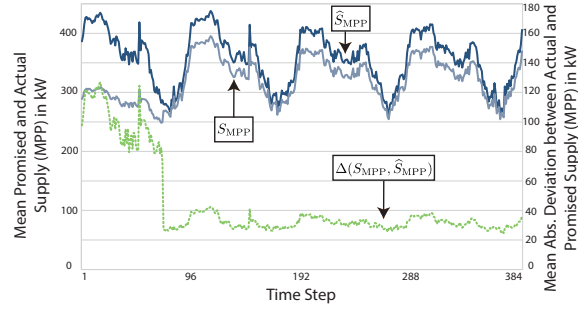


Figure 10.5: The promised \hat{S}_{MPP} and actual S_{MPP} supply of the MPP in E-nt. The mean absolute deviation $\Delta(S_{MPP}, \hat{S}_{MPP})$ between S_{MPP} and \hat{S}_{MPP} mirrors the MPP's varying and contribution-specific behavior.

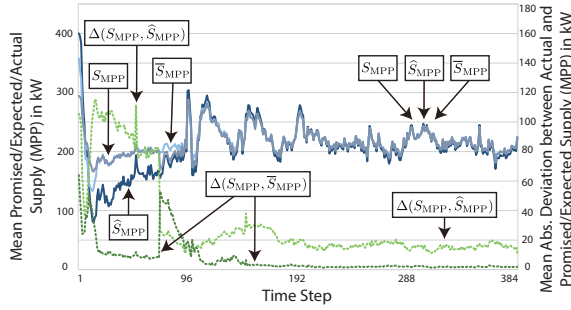


Figure 10.6: The promised \hat{S}_{MPP} , expected \bar{S}_{MPP} , and actual S_{MPP} supply of the MPP in E-t. As in Figure 10.5, the mean absolute deviation $\Delta(S_{MPP}, \hat{S}_{MPP})$ between S_{MPP} and \hat{S}_{MPP} mirrors the MPP's varying behavior. The fact that the mean absolute deviation $\Delta(S_{MPP}, \bar{S}_{MPP})$ between S_{MPP} and \bar{S}_{MPP} is smaller than $\Delta(S_{MPP}, \hat{S}_{MPP})$ confirms that TruCAOS's trust-based decisions mitigate epistemic uncertainties.

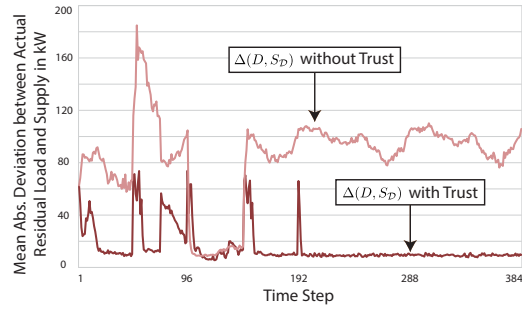


Figure 10.7: The mean absolute deviation $\Delta(D, S_V)$ between the actual residual load D and the total actual supply S_V in E-nt compared to E-t. TruCAOS's trust-based decisions significantly decrease $\Delta(D, S_V)$.

supply depends on \hat{S}_{MPP} . The MPP's behavior causes an average $\Delta(S_{MPP}, \hat{S}_{MPP})$ of 44.25 kW with $\sigma = 27.55$ kW.

As for the evaluation scenario E-t, Figure 10.6 depicts how the AVPP allocates resources of the MPP in case trust-based decisions are made. The AVPP needs approximately 18 time steps until the trust value accurately reflects the MPP's behavior. After these time steps, TruCAOS's trust-based principle allows the AVPP to significantly reduce the deviation between the MPP's expected \bar{S}_{MPP} and actual S_{MPP} supply. The MPP's behavior causes an average absolute deviation $\Delta(S_{MPP}, \bar{S}_{MPP})$ of only 7.34 kW with $\sigma = 10.77$ kW. At time step 72, the MPP changes its behavior, which is reflected in the peak of $\Delta(S_{MPP}, \bar{S}_{MPP})$. About 20 time steps are needed to adapt to the new behavior. During these time steps, \hat{S}_{MPP} is actually more accurate than \bar{S}_{MPP} . However, starting at time step 144, the curves of S_{MPP} , \hat{S}_{MPP} , and \bar{S}_{MPP} almost coincide. The MPP's promised and actual contributions are thus very different from those it makes in the evaluation scenario E-nt in which the AVPP does not employ the

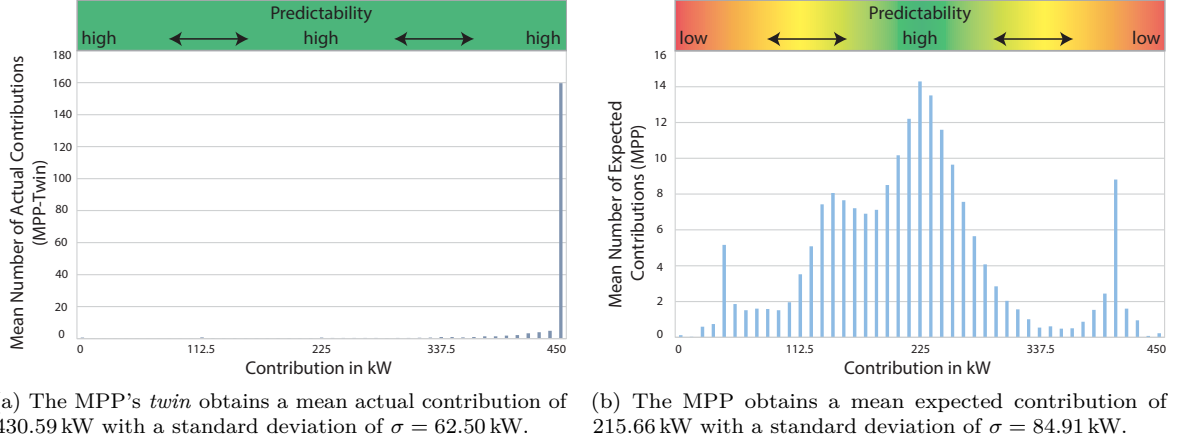


Figure 10.8: The number of actual contributions of the MPP's twin (see Figure 10.8a) and the number of expected contributions of the MPP (see Figure 10.8b) for time steps $t \in [192, 384]$. The width of the intervals corresponds to 10 kW. The MPP's sweet spot of high predictability can be found at 225 kW. The MPP's twin always complies with its scheduled supply and is thus very predictable.

concept of predictability (see Figure 10.5). This is mainly because, in the evaluation scenario E-t, the risk-averse AVPP allocates the resources of the MPP in such a way that \hat{S}_{MPP} is near its sweet spot of high predictability at 225 kW (see Figure 10.6). Recall that, at its sweet spot, the MPP does not deviate from its promises and is thus very predictable. Hence, the mean values of S_{MPP} , \hat{S}_{MPP} , and \bar{S}_{MPP} are approximately equal (the average of $\Delta(S_{\text{MPP}}, \hat{S}_{\text{MPP}})$ is 31.43 kW with $\sigma = 28.93$ kW). As a result, the expected supply \bar{S}_{MPP} is 76.65 % more accurate than the promised supply \hat{S}_{MPP} . In comparison to E-nt, uncertainties in promised supply – measured as the average of $\Delta(S_{\text{MPP}}, \hat{S}_{\text{MPP}})$ – could be reduced by 28.97 %. This behavior is also visualized in the histograms in Figure 10.8: The MPP's *twin* is almost always allowed to contribute with its maximum capacity (see Figure 10.8a). This corresponds to the curve of the MPP's actual supply S_{MPP} for the evaluation scenario E-nt in Figure 10.5. As opposed to that, the AVPP's trust-based and risk-averse decisions shift the contributions of the MPP into the region around its sweet spot in E-t (see Figure 10.8b). This shows that TruCAOS actively and effectively mitigates the presence of aleatoric uncertainties by accepting proposals in a way that allows intermediaries to increase the dispatchable agents' predictability. Because the contributions of the MPP are much lower than those of its twin, TruCAOS's trust-based decisions further incentivize dispatchable agents to behave benevolently. In electric power systems, this means that a power plant's operator is incentivized to ensure that it can comply with its schedule (see Chapter 13).

Figure 10.7 summarizes our observations: By using trust-based techniques in E-t, the average of the mean absolute deviation $\Delta(D, S_V)$ between the actual residual load D and the total actual supply S_V is only 15.74 kW ($\sigma = 14.24$ kW), compared to 84.27 kW ($\sigma = 31.76$ kW) without trust-based measures in E-nt. As for E-t, this results in an average total violation of the residual load of 6044.55 kW per run. This is 81.32 % lower than in case of E-nt, where we have a total violation of the residual load of 32360.15 kW. On average, TruCAOS terminated after 5.58 ($\sigma = 0.44$) bidding iterations in E-nt and 5.55 ($\sigma = 0.40$) iterations in E-t. Hence, trust-based measures did not have a significant influence on the number of bidding iterations.

To assess TruCAOS's runtime performance and the quality of created schedules in this setting, we use an additional evaluation scenario without any uncertainties originating from the dispatchable power plants' supply or residual load predictions. For comparison, we employ the regio-central approach *RegioC*. In this additional evaluation scenario, TruCAOS as well as *RegioC* are able to satisfy the entire residual load in all time steps (this corresponds to the scheduling problem's primary objective). *RegioC* needs 260.10 ms ($\sigma = 66.42$ ms) on average to create schedules for all dispatchable power plants. By

incorporating exclusively cheap hydro power plants (this is possible because their aggregated maximum output exceeds the maximum residual load), RegioC obtains optimal average costs of $6.50 \frac{\text{euro cent}}{\text{kWh}}$ ($\sigma = 0.0 \frac{\text{euro cent}}{\text{kWh}}$). In this evaluation scenario, TruCAOS also achieves optimal costs but needs only an average of 57.13 ms ($\sigma = 15.23$ ms) to create schedules. Consequently, TruCAOS is 78.04 % faster than RegioC. Please note that, in the *worst case*, the algorithms could choose an allocation of resources that perfectly satisfies the residual load but causes average costs of $17.50 \frac{\text{euro cent}}{\text{kWh}}$, which are 169.23 % higher than those found by TruCAOS and RegioC.

Although we only regarded a single misbehaving dispatchable agent in our empirical analysis, we expect intermediaries to behave in a very similar way in the presence of multiple misbehaving dispatchable agents. We hypothesize that intermediaries shift the contributions of misbehaving dispatchable agents into ranges in which they are more predictable as long as a sufficient amount of trustworthy dispatchable agents is available that, together, are able to compensate for the adjustments made with respect to the misbehaving dispatchable agents' supply. In this context, the factor κ_v that maps changes in predictability to costs can be used to steer the intermediaries' attitude towards risk (see Section 10.4).

Analysis of the Runtime Performance and Schedule Quality in Hierarchical Systems

In the second half of our evaluation, we investigate the impact of different hierarchies of AVPPs on TruCAOS's performance in terms of scheduling times and the schedules' quality. Again, the regio-central approach RegioC serves as benchmark. Note that we refrain from regarding uncertainties in this evaluation setting, i.e., all power plants and consumers adhere to their promises and predictions. As for uncertain supply, that is because it is far from obvious how to implement effective trust-based measures in RegioC's optimization problems so that they can be *efficiently* solved by CPLEX. We basically would have to encode the complete trust model for measuring uncertain supply, which we presented in Section 10.3, in CPLEX's optimization programming language (OPL). Especially the non-linear and non-monotonic characteristic of the function mapping a dispatchable agent's scheduled supply to its expected supply (i.e., increasing a power plant's scheduled output could actually decrease its expected output) complicates CPLEX's search for high-quality solutions. This problem does not exist in case of TruCAOS since intermediaries only have to choose from a set of received proposals. Regarding uncertain demand, RegioC can employ the same trust model as TruCAOS. For the comparison of the two algorithms, it thus makes no difference whether schedules are created for the predicted or the expected demand.

Test Bed

To investigate the influence of hierarchical system structures on TruCAOS's and RegioC's performance, we performed our evaluations on four different structures called "superflat", "flat", "deep", and "deeper" of height 1, 2, 3, and 5, and with an average number of 173, 14.23, 6.55, and 2.42 dispatchable power plants per AVPP, respectively. All these structures were generated offline on the basis of the same set of 173 dispatchable and 350 non-dispatchable power plants. The "superflat" structure corresponds to a centralized system and serves to evaluate the algorithms' ability to create schedules for a large group of dispatchable agents. In Section 9.2, a hierarchy of height 2 allowed RegioC to obtain an adequate compromise between runtime performance and solution quality, which is why we additionally consider the "flat" structure of height 2 in this evaluation. The purpose of the structures "deep" and "deeper" is to examine the effect of deeper hierarchies on the algorithms' behavior. In our experiments, we disabled the system's ability to change these structures to be able to analyze their influence on the algorithms' performance. In each of the 200 simulation runs, the dispatchable power plants have to satisfy a prescribed residual load over a period of one day, which corresponds to $|\mathcal{T}| = 96$ time steps.

In a nutshell, we examine the following three questions of interest: (1) *Does TruCAOS outperform RegioC in terms of runtime performance?* (2) *Does TruCAOS obtain better results w.r.t. demand satisfaction and costs?* (3) *How does the hierarchical system structure affect the schedules' quality and runtime performance?*

	superflat		flat		deep		deeper	
	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS
#Dispatchable Power Plants per AVPP	173.00		14.23		6.55		2.42	
Max. Seq. Scheduling Time per Time Step [ms]	13612.19 (6264.49)	228.89 (59.87)	405.96 (139.57)	369.08 (645.09)	412.03 (143.70)	510.18 (321.91)	467.26 (128.86)	740.50 (464.30)
Scheduling Time per AVPP and Time Step [ms]	13612.19 (6264.49)	228.89 (59.87)	131.35 (70.38)	46.29 (5.08)	72.30 (49.25)	33.14 (2.70)	40.17 (33.48)	18.35 (1.79)
#Bidding Iterations per AVPP and Time Step	–	3.97 (0.66)	–	2.49 (0.53)	–	3.01 (0.53)	–	1.88 (0.51)
#Detected Local Optima per Time Step	–	0.00 (0.00)	–	0.18 (0.04)	–	2.96 (0.20)	–	14.24 (1.31)
Total Violation of the Residual Load per Run [kW]	0.00 (0.00)	2.40 (0.02)	0.00 (0.00)	6.98 (0.05)	926.97 (479.21)	80.99 (2.75)	2905.38 (1144.81)	140.93 (1.44)
Unit Price of Electricity per Time Step [euro cent/kWh]	10.32 (1.11)	10.32 (1.11)	11.22 (0.74)	11.46 (0.91)	12.07 (0.49)	12.69 (0.72)	12.71 (0.29)	12.69 (0.61)

Table 10.1: Evaluation results obtained by TruCAOS and RegioC for different hierarchies of AVPPs. All results are averaged over the 200 runs per evaluation scenario. Values in parentheses denote standard deviations.

Results

Table 10.1 shows the results obtained for the different hierarchical structures. Unsurprisingly, RegioC struggles with the large number of power plants schedules have to be created for in the “superflat” scenario. While RegioC requires an average of 13612.19 ms to create valid schedules, TruCAOS only needs remarkable 228.89 ms on average. This is 98.32 % faster. As discussed in Section 10.1, that is because TruCAOS distributes the computational complexity of schedule creation among all participating agents. In particular, dispatchable power plants create proposals for their schedules themselves. Beyond that, TruCAOS obtains a lower coefficient of variation of scheduling times than RegioC ($\frac{59.87}{228.89} \approx 0.26$ compared to $\frac{6264.49}{13612.19} \approx 0.46$), meaning that TruCAOS’s scheduling times are about twice as stable as those of RegioC. Further, TruCAOS is able to satisfy the demand at the same *optimal* costs of $10.32 \frac{\text{euro cent}}{\text{kWh}}$ as RegioC. Please note that the cheapest dispatchable power plants have production costs of $6.50 \frac{\text{euro cent}}{\text{kWh}}$ and that, in the *worst case*, the algorithms could opt for an allocation that accurately satisfies the residual load but leads to a unit price of electricity of $17.33 \frac{\text{euro cent}}{\text{kWh}}$. Under these circumstances, TruCAOS’s results are quite remarkable.

RegioC significantly benefits from the scheduling problem’s decomposition in case of the “flat” hierarchy. The maximum sequential scheduling time (i.e., the longest of all serial paths that result from adding the scheduling times for each branch in the hierarchy; see Equation (6.1)) drops to 405.96 ms and an AVPP only needs 131.35 ms on average to calculate schedules. We can explain this by the lower number of power plants schedules have to be created for (14.23 on average compared to 173 in case of the “superflat” structure). On the other hand, the costs slightly increase by 8.72 % due to abstraction and indirections. While the mean scheduling time per AVPP significantly decreases to 46.29 ms for TruCAOS, we see that the mean max. seq. scheduling time increases by 61.25 % compared to the “superflat” hierarchy. We explain this behavior by three observations: First, due to the complexity of the AVPPs’ control models, AVPPs need more time for generating proposals than physical power plants. Second, additional hierarchy layers introduce additional overhead: Now, multiple AVPPs have to solve the combinatorial auction problem, and, depending on their relative position in the hierarchy, these problems have to be solved sequentially, thereby adding to the max. seq. scheduling time. Top-down dependencies also restrict the parallel creation of proposed schedules to specific subsets of dispatchable power plants. Third, intermediaries require a larger number of schedule corrections to meet the residual load in systems consisting of smaller AVPPs. These corrections add to the max. seq. scheduling time. However, although TruCAOS’s max. seq. scheduling time increases, it is still *shorter* than RegioC’s. The costs TruCAOS obtains are also still comparable to those of RegioC.

With regard to TruCAOS, the mean scheduling time per AVPP further decreases – whereas the max. seq. scheduling time continues to grow – when we further decompose the scheduling problem by means of the structures “deep” and “deeper” (see Table 10.1). TruCAOS’s costs exceed those of RegioC in

case of the “deep” structure but are comparable to the costs of RegioC in case of the structure “deeper”. Since RegioC’s max. seq. scheduling times also increase in these scenarios, we can conclude that – with respect to RegioC – the hierarchy “flat” is a sweet spot where RegioC achieves low max. seq. scheduling times at adequate costs. As for TruCAOS, the sweet spot is the hierarchy “superflat” which allows the algorithm to obtain lower costs and shorter max. seq. scheduling times than RegioC in the “flat” hierarchy. Both algorithms satisfy the residual load very accurately for these structures.

The total violation of the residual load grows considerably with the height of the hierarchy, though. As discussed and investigated in Part III, this can be attributed to an increase in fragmentation and resulting abstraction errors. Because increasing fragmentation reduces the AVPPs’ degrees of freedom, this phenomenon manifests in a considerable increase in the number of detected local optima by TruCAOS (cf. hierarchies “deep” and “deeper”). As for RegioC, the total violation of the residual load reaches high values of 2905.38 kW for the hierarchy “deeper”. In comparison, TruCAOS achieves relatively low deviations of only 140.93 kW for the same hierarchy. We ascribe TruCAOS’s ability to deal with fragmentation and abstraction errors to its principle of schedule corrections (see Section 10.1). These allow the AVPPs to iteratively compensate for deviations between scheduled supply and assigned residual load during the schedule creation. On the flip side, this leads to longer max. seq. scheduling times.

Note that these observation are in tune with those we made in the evaluation of the self-organized formation of scalable hierarchical system structures in Section 9.2. There, we already observed that deeper hierarchies consisting of a higher number of AVPPs do not necessarily lead to shorter max. seq. scheduling times but come at the expense of solution quality.

To summarize, TruCAOS allows for the creation of schedules for a large number of dispatchable agents in short time. Even in large systems, the agents can thus self-organize into relatively flat hierarchies. Because this characteristic avoids fragmentation and increases the system structures’ stability (see Section 9.2), it is also advantageous with regard to the quality of created schedules in terms of the violation of the demand and costs.

Chapter Summary and Outlook

In this chapter, we presented an auction- and trust-based mechanism, called *TruCAOS*, that solves the scheduling problem in hierarchical systems in a cooperative and regionalized manner. Instead of assigning schedules to subordinates, an intermediary acts in the role of an auctioneer that redistributes its fraction of the overall demand in an iterative and incremental process. Subordinate dispatchable agents actively participate in the process of schedule creation by making suggestions of how they could adapt their schedules to increase the demand’s satisfaction. Because this principle reduces the scheduling problem’s complexity, TruCAOS allows the agents to self-organize into flatter hierarchies consisting of larger subsystems than the regio-central approach, which prevents the introduction of unnecessary abstraction errors and improves the system structures’ stability. Our evaluations in Sections 9.2 and 10.5 confirm that this is advantageous to the schedules’ quality in terms of demand satisfaction and costs. TruCAOS is therefore ideally suited for creating high-quality schedules in large scale systems in short time. To deal with uncertain supply and demand, TruCAOS augments the process of schedule creation by several trust-based measures that address both unavoidable as well as avoidable misbehavior. We demonstrated that TruCAOS reduces the influence of systematic uncertainties in demand predictions and promised supply by forming expectations of the agents’ future behavior. Moreover, TruCAOS mitigates the presence of aleatoric uncertainties by accepting proposals in a way that allows intermediaries to put dispatchable agents into states of higher predictability. This is obtained by decreasing the price-performance ratio of agents that propose to make contributions in ranges of low predictability. Although this measure primarily addresses unavoidable misbehavior, trustworthy dispatchable agents can charge higher prices for a contribution than untrustworthy agents, which results in price premiums and price discounts. In conjunction with a payment function that distributes rewards according to the agents’ actual instead of their scheduled supply and that penalizes schedule violations, the measures thus also incentivize benevolent behavior.

In future work, we will investigate how and to what degree we can integrate the dispatchable agents' individual preferences, e.g., expressed in the form of *constraint relationships* [188], into the schedule creation. To this end, we have to extend the optimization problems solved for proposal generation and for identifying suitable winner proposals in a way that takes both individual as well as organizational objectives into account.

In the next chapter, we enhance TruCAOS by the ability to create schedules for different possible development of the demand. Based on the concept of Trust-Based Scenario Trees (see Chapter 5), this allows the system to cope with volatile demand that follows different behavioral patterns. In Chapter 12, we show how agents utilize these schedules to reactively compensate for deviations between the actual demand and supply at runtime without having to recalculate schedules.

Robust Solutions to the Scheduling Problem on the Basis of Trust-Based Scenario Trees

Summary. One way to deal with volatile demand that follows different behavioral patterns is to take several scenarios for its development into account. To this end, we adopt techniques from the field of *online stochastic optimization* [83] in order to find robust solutions to the scheduling problem. By utilizing the concept of *Trust-Based Scenario Trees* (TBSTs) presented in Chapter 5, intermediaries can create a probabilistic model of such volatile demand at runtime. Creating schedules on the basis of TBSTs allows intermediaries (1) to anticipate aleatoric, i.e., intrinsic random and irreducible, uncertainties by considering multiple scenarios, and (2) to reduce epistemic, i.e., systematic, uncertainties by putting more effort into the optimization of likely scenarios. By taking the trees' structure into account, the agents self-improve their flexibility in adapting to the actual demand as they can switch to the most appropriate schedule at runtime. To support such switches and to strengthen the agents' ability to meet the demand even if it does not correspond to one of the considered scenarios, we enable the agents to schedule an appropriate amount of degrees of freedom, i.e., reserves. These are used for reactive supply adjustments. To cope with the complexity of the extended scheduling problem, we show how TBST-based schedules can be created in hierarchical systems. Furthermore, we enhance the auction-based scheduling mechanism TruCAOS for the creation of TBST-based schedules and the provision of reserves. To avoid that deviations between supply and demand propagate through the system, each intermediary takes responsibility for scheduling its subordinate dispatchable agents in a way that allows them to compensate for such deviations locally, i.e., in the intermediary's subsystem. Our evaluations reveal the advantage of anticipating prediction errors locally and demonstrate that TruCAOS clearly outperforms the regio-central approach when searching for robust solutions to the scheduling problem.

Publication. The concepts and results outlined in this chapter have been published in Anders et al. [11] and Kosak et al. [123].

As stated before, a major challenge is to create schedules that enable the dispatchable agents to satisfy the demand despite inaccurate predictions, fluctuations, and unexpected events. In terms of the residual load, fluctuations originate from changing weather conditions and stochastic consumer behavior, among others. For the system's stable and efficient operation, the agents have to quantify and anticipate these uncertainties and to incorporate this information into their scheduling decisions. While Chapter 10 primarily focused on how to deal with an uncertain provision of scheduled or promised resources, this chapter focuses on uncertain volatile demand that follows different behavioral patterns.

In Chapter 5, we presented the concept of *Trust-Based Scenario Trees* (TBSTs). A TBST is a probabilistic model that approximates the stochastic process governing the behavior of a single or a group of non-dispatchable agents. It consists of a number of *Trust-Based Scenarios* (TBSs), each

indicating an expectation, i.e., a possibility, of how the agents' behavior might develop in a sequence of future time steps. Furthermore, each TBS has a certain probability of occurrence so that the complete TBST describes an empirical probability mass function specifying a discrete probability distribution over multiple time steps. As we demonstrated in Section 5.3, this allows TBSTs to capture volatile behavior that follows different behavioral patterns.

With regard to the resource allocation problem, each intermediary uses TBSTs to quantify deviations from its *local* demand predictions and to deduce expected deviations. Recall that an intermediary's local demand is the aggregate of the demand resulting from its subordinate non-dispatchable agents and intermediaries. As explained in Section 5.2, creating a single TBST for the group of its subordinate non-dispatchable agents allows an intermediary to capture statistical relationships between the agents' behavior. This might be necessary to derive adequate expected deviations (e.g., think of weather predictions for adjacent regions or consumer attitude). The corresponding *deviation tree* is composed of *deviation scenarios*, each representing a possible sequence of deviations from a demand prediction. By adding a given predicted demand to each deviation scenario, an intermediary derives a so-called *demand tree* that consists of *demand scenarios* (see Section 5.2). A demand tree represents different expected developments of the demand.

If intermediaries created schedules for a single demand scenario, they would have to decide for a specific TBS, such as the *anticipated scenario* (i.e., the most likely development of the demand) or the riskiest demand scenario with regard to a certain measure. Since this decision can turn out to be wrong, it is rather risky for an intermediary and the whole system to rely on a single demand scenario. That is because the dispatchable agents might not be able to change their supply quickly enough to reactively adapt to the actual demand. A risk-averse intermediary would be well-advised to take several demand scenarios as well as their probabilities into account. In the literature, one can find different approaches that take account of multiple scenarios when solving an optimization problem under uncertainty (Sahinidis [183] surveys numerous methods for optimization under uncertainty). In terms of our scheduling problem, these approaches presume that reducing the risk of not being able to deal with the actual demand, i.e., its *realization*, outweighs the higher unit price of supply that probably results from considering multiple scenarios.

Robust optimization (cf. [32, 134]) as known in *mathematical optimization* (not to be confused with the general term of "robust optimization" as used in this thesis), for instance, assumes that uncertain parameters, such as the demand, can be either restricted to a certain interval or a set of scenarios. Uncertainties might further be subject to the variable assignment. Usually, robust optimization does not assume that a probability distribution of the uncertain parameters is known. For such situations, it proposes to solve a minimax optimization problem whose objective is to minimize the costs that result from the (possibly assignment-specific) worst-case scenario. With regard to the power management case study, one could schedule a group of power plants in a way that minimizes the maximum amount of expensive reserves that have to be activated by the primary control system in order to meet another residual load scenario. The residual load scenario maximizing these expenses corresponds to the worst case scenario for the created schedules.

Another well-known framework for optimization problems in uncertain environments is *stochastic programming* [198]. In contrast to robust optimization, stochastic programming assumes that a discrete probability distribution of the uncertain parameters is available, e.g., in the form of a scenario tree. Based on this knowledge, stochastic programming suggests to minimize the expected value of a chosen cost function. Applied to power systems, the idea is, e.g., to minimize the expected deviation between production and consumption. As depicted in Figure 11.1, this principle yields *robust solutions* as the effect of changes in the environment or the decision variables is less severe than in a situation in which only a single scenario is regarded [42].¹ We thus refer to robust solutions to the scheduling problem as *robust schedules*.

Indeed, TBSTs specify a discrete probability distribution over multiple time steps so that we can make use of the principle of stochastic programming. By taking the scenarios' probabilities into account,

¹According to Branke [42], another possibility for obtaining robust solutions would be to maximize the probability of keeping the value of a cost function (e.g., the violation of the residual load) below a certain threshold.

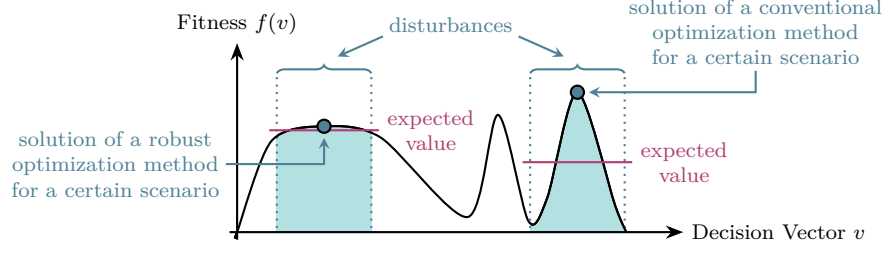


Figure 11.1: A fitness landscape illustrating the idea of robust solutions (according to [42]): A conventional optimization method searches for the optimal solution that might only be feasible for a specific scenario (e.g., it finds the least expensive schedules for a residual load of 10 MW). In case of disturbances, the solution's actual fitness might turn out to be much lower, though (e.g., assume that the actual residual load is 12 MW so that a costly peaking power plant has to be ramped up to satisfy the additional demand). A robust optimization method incorporates uncertainties in the form of multiple scenarios. Robustness is achieved by, for instance, searching for a plateau of good fitness values that yields a higher expected fitness than the solution of the conventional optimization method (e.g., as a residual load of either 10 MW or 12 MW is expected, it schedules the power plants in such a way that the residual load can be satisfied with inexpensive power plants in both scenarios).

intermediaries create schedules that minimize the *expected* violation of the demand. As we explain in Section 11.1, each intermediary creates schedules for the entire demand tree and thus solves a so-called *multi-stage stochastic program*. Due to the computational complexity of multi-stage stochastic programs [197], most of the scenario-based approaches that can be found in the literature (e.g., [178, 193]) use a *two-stage stochastic program* as approximation. A two-stage stochastic program has the following form [198]:

$$\begin{aligned}
& \underset{\mathbf{x}, \mathbf{y}_\omega}{\text{minimize}} && \mathbf{c}^T \mathbf{x} + \sum_{\omega \in \Omega} p_\omega \cdot (\mathbf{d}^T \mathbf{y}_\omega) \\
& \text{subject to} && \mathbf{A} \mathbf{x} \leq \mathbf{b} \\
& && \mathbf{B}_\omega \mathbf{x} + \mathbf{C}_\omega \mathbf{y}_\omega \leq \mathbf{e}_\omega, \forall \omega \in \Omega \\
& && \mathbf{x} \geq \mathbf{0}, \mathbf{y}_\omega \geq \mathbf{0}, \forall \omega \in \Omega
\end{aligned}$$

A solution to a two-stage problem is composed of a first- and a second-stage decision. Applied to our scheduling problem, the first-stage decision \mathbf{x} states how many resources the agents should provide independently of the realization of the uncertain demand that has to be met in $t_{\text{now}} + \Delta\tau$ (recall that we defined $\Delta\tau$ as a schedule's resolution in Section 1.2). Feasible assignments for \mathbf{x} are subject to the linear constraints $\mathbf{A} \mathbf{x} \leq \mathbf{b}$. For instance, if the vector \mathbf{x} defines the dispatchable agents' scheduled supply, the product of the identity matrix \mathbf{A} and \mathbf{x} must not exceed the agents' maximal supply stated in the vector \mathbf{b} . \mathbf{x} prescribes a partial solution that is well positioned for all possible scenarios (e.g., dispatchable power plant a should provide 10 MW). The second-stage decision \mathbf{y}_ω serves as a scenario-specific corrective, i.e., a *recourse action*, that is carried out to balance supply and demand as soon as its realization is known (e.g., in case of scenario ω' , the power plant a should provide an extra 5 MW). A typical example would be to take the risk of buying resources at a higher price once the actual demand is known instead of pre-ordering them due to storage costs and the possibility of a surplus. For each scenario $\omega \in \Omega$ with probability p_ω , the vector \mathbf{y}_ω has to be chosen in a way that the scenario-specific constraints $\mathbf{B}_\omega \mathbf{x} + \mathbf{C}_\omega \mathbf{y}_\omega \leq \mathbf{e}_\omega$ are satisfied (\mathbf{B}_ω and \mathbf{C}_ω are scenario-specific matrices and \mathbf{e}_ω is a scenario-specific vector that holds, e.g., the expected demand in scenario ω). A solution to a specific demand scenario ω is thus a combination of \mathbf{x} and the recourse action \mathbf{y}_ω . For instance, in case of scenario ω' , power plant a has to provide $10 \text{ MW} + 5 \text{ MW} = 15 \text{ MW}$.

The concept of recourse actions (which is also used in multi-stage problems) is, however, not applicable to our resource allocation problem because inertia is likely to prevent the dispatchable agents from performing the action as scheduled. For example, because of its limited rate of change, power plant a

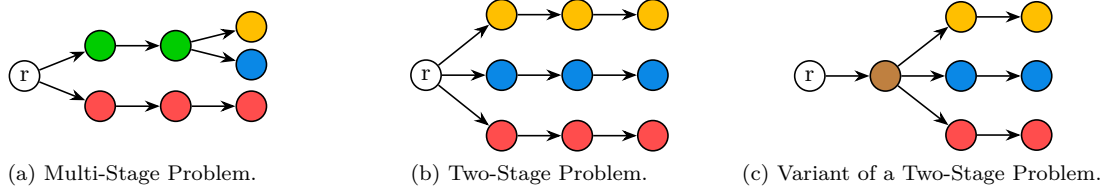


Figure 11.2: Different ways to consider and deal with the scenarios contained in a scenario tree (scenarios are color-coded): While multi-stage problems take account of the tree structure (see Figure 11.2a), two-stage problems consider the same scenarios in the form of a fan of unrelated scenarios (see Figure 11.2b). Figure 11.2c shows a variant of two-stage problems presented in [193]. This variant abolishes recourse actions for the next relevant time step (cf. the root’s child) by representing the uncertainties by a single expected value. The expected value is the average of the root’s children in Figure 11.2a. Evidently, regarding an expected value does not allow for encoding strategies for how to react to different situations.

cannot instantly provide the extra 5 MW. As opposed to this, the dispatchable agents have to monitor the development of the demand and to make reactive supply adjustments in order to maintain the balance as well as possible (see Chapter 12).

Apart from inertia, permanent reactive adjustments are necessary because of the following three reasons: **(R1)** The scheduling problem’s complexity and the uncertainties involved call for rather coarse-grained schedules, which is why we create them for time steps that are multiples of $\Delta\tau$. Still, the demand has to be satisfied in a fine-grained time pattern of $\Delta t \leq \Delta\tau$ (see Section 1.2). In the synchronous grid of Continental Europe, for instance, schedules are typically created with a resolution of $\Delta\tau = 15$ min, whereas imbalances have to be detected and compensated for within seconds (e.g., $\Delta t = 5$ s) to ensure the grid’s stable operation [214]. **(R2)** Some agents might contribute according to the wrong scenario. Such a mistake is caused by a false expectation of how the demand will develop, e.g., from t_{now} until $t_{\text{now}} + \Delta\tau$. For instance, the agents might contribute according to the anticipated scenario, but a less probable scenario actually occurs. **(R3)** The system must be able to deal with unforeseen developments of the demand, i.e., even those that are not captured by any demand scenario in the demand tree.

In the context of permanent reactive adjustments, considering a multi-stage instead of a two-stage problem is advantageous because scenarios are treated as a tree instead of a fan of unrelated scenarios (see Figure 11.2).² By taking account of the demand tree’s structure (this means that we schedule contributions for nodes, which might be part of multiple scenarios), we ensure that the agents cannot only provide the scheduled output in $t_{\text{now}} + \Delta\tau$ but also in subsequent time steps (note that there might be a branch following a node in $t_{\text{now}} + \Delta\tau$). This characteristic allows the agents to delay their decision which demand scenario to choose, i.e., their assumption in which direction the demand develops, until the time step a branch in the tree has to be taken. In this context, TBST-based schedules encode strategies that indicate how many resources dispatchable agents have to provide in which situation. In terms of stochastic programming, they constitute a decision rule [198]. In Chapter 12, we show that TBST-based schedules proactively guide permanent reactive adjustments, i.e., self-adaptation, of inert agents, thereby improving the system’s efficiency, stability, and robustness. To strengthen the dispatchable agents’ ability to deal with unforeseen situations (i.e., aleatoric uncertainties) by means of reactive supply adjustments, we additionally enable them to increase their available degrees of freedom by scheduling feedback-driven *reserves*.

All these aspects, anticipating uncertainties, creating schedules on the basis of demand trees that have been learned online, and incorporating reserves aim at proactively increasing the system’s robustness with respect to epistemic and aleatoric uncertainties. In terms of the two dimensions of robustness introduced in Section 1.4, these aspects proactively increase the system’s ability to remain in acceptable states and thus to maintain its functionality despite detrimental influences (first dimension), and

²In stochastic programming, this is obtained by so-called *non-anticipativity constraints*.

to return into an acceptable state after a disturbance occurred that caused the system to leave the acceptance space (second dimension). Our approach to deal with the computational complexity of solving multi-stage stochastic optimization problems and scheduling reserves is (1) to make use of the self-organizing hierarchical system structure to decompose the overall scheduling problem, and (2) to solve the scheduling problem within each organization (i.e., AVPP) by means of an extended version of the auction-based scheduling mechanism TruCAOS presented in Chapter 10. As stated before, the environment’s dynamic and uncertain nature justifies the application of these heuristics since they enable the system to update schedules more frequently on the basis of up-to-date data than a method that guarantees optimal solutions at the expense of much higher runtimes.

The remainder of this chapter is structured as follows: In Section 11.1, we formulate the scheduling problem defined in Section 2.2 on the basis of TBSTs and the idea of *online stochastic optimization* [83], which combines online algorithms and stochastic programming. Afterwards, we outline the top-down TBST-based schedule creation in hierarchical systems in Section 11.2. The presented approach allows each subsystem to compensate for deviations between supply and demand locally, i.e., right where they occur. This prevents their propagation through the system. An empirical evaluation at the end of Section 11.2 demonstrates the advantage of using TBSTs in the context of the scheduling problem, and the benefit of enabling each intermediary to anticipate and quantify inaccuracies in its individual *local* demand predictions. Subsequently, we augment the TBST-based scheduling problem by the provision of feedback-driven reserves in Section 11.3. This improves the dispatchable agents’ ability to reactively compensate for deviations between supply and demand without having to recalculate schedules. Sections 11.1 to 11.3 explain the top-down creation of TBST-based schedules and the provision of reserves in hierarchical system structures from the regio-central perspective. For finding high-quality solutions to the enhanced scheduling problem in large-scale hierarchical systems in short time, we extend TruCAOS by the ability to create TBST-based schedules and to schedule reserves in Section 11.4. Our empirical evaluation provided in Section 11.5 shows that TruCAOS significantly outperforms the regio-central approach that is based on a state-of-the-art optimizer.

Related work in the context of algorithms for scenario-based resource allocation and the provision of reserves is discussed in Chapter 13. In Chapter 12, we describe how TBST-based schedules proactively guide the dispatchable agents’ permanent reactive supply adjustments.

11.1 Formalization of the Scenario-Based Scheduling Problem

In the following, we describe the creation of TBST-based schedules from the perspective of an arbitrary intermediary $\lambda \in \mathcal{I}$, taken from somewhere in the hierarchy. As before, λ creates schedules for its subordinate dispatchable agents \mathcal{V}_λ for the scheduling window $\mathcal{W} = \{t_{\text{now}} + i \cdot \Delta\tau \leq t_{\text{now}} + H \mid i \in \mathbb{N}_{\geq 1}\}$. The main difference between the scheduling problem as formulated in Equation (2.1) and the TBST-based reformulation is that λ creates schedules according to the demand tree $D_\lambda^{\mathfrak{T},dis}$ instead of the expected demand $\overline{D}_\lambda^{\mathcal{W}}$ (or the demand prediction $\widehat{D}_\lambda^{\mathcal{W}}$). The demand tree $D_\lambda^{\mathfrak{T},dis}$ specifies the demand λ has to *distribute* to its subordinate dispatchable agents \mathcal{V}_λ . For now, we assume $D_\lambda^{\mathfrak{T},dis}$ to be given (we abandon this assumption in Section 11.2). Further, we assume that $D_\lambda^{\mathfrak{T},dis}$ is based on λ ’s local deviation tree $\delta_\lambda^{\mathfrak{T}}$ so that $D_\lambda^{\mathfrak{T},dis}$ captures the uncertainties in λ ’s local demand predictions. Consequently, both trees have the same shape. Every time schedules are calculated, λ updates $\delta_\lambda^{\mathfrak{T}}$ on the basis of its latest experiences.

When calculating schedules, λ now stipulates the supply $S_a^{\mathfrak{T}}[n]$ of a subordinate dispatchable agent $a \in \mathcal{V}_\lambda$ for each node $n \in D_\lambda^{\mathfrak{T},dis} \setminus \{r\}$. The schedule $S_a^{\mathfrak{T}}$ of an agent a is thus a tree (indicated by the superscript “ \mathfrak{T} ”) whose root reflects a ’s current state, which includes its current supply. This is in contrast to the former definition of the scheduling problem, where a dispatchable agent’s schedule S_a was a sequence of scheduled contributions $S_a[t]$, one for each time step $t \in \mathcal{W}$. With regard to an agent’s schedule $S_a^{\mathfrak{T}}$ and the demand tree $D_\lambda^{\mathfrak{T},dis}$, each node $n \in D_\lambda^{\mathfrak{T},dis}$ is associated with exactly one node $n' \in S_a^{\mathfrak{T}}$ so that a demand $D_\lambda^{\mathfrak{T},dis}[n]$ unambiguously identifies the corresponding scheduled supply $S_a^{\mathfrak{T}}[n']$ (therefore, we also write $S_a^{\mathfrak{T}}[n]$ instead of $S_a^{\mathfrak{T}}[n']$). Further, each node in $S_a^{\mathfrak{T}}$ is associated with at least one node in $D_\lambda^{\mathfrak{T},dis}$. These characteristics allow λ to encode strategies for supply adjustments needed to

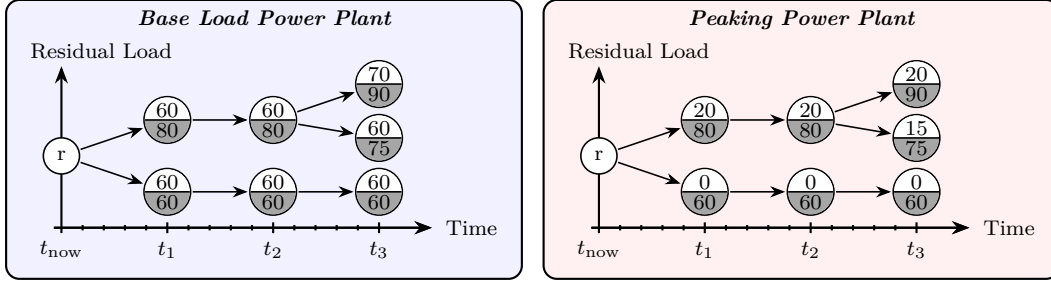


Figure 11.3: The power plants use TBST-based schedules as a blueprint for how much power to provide in which situation. For each node, the sum of the scheduled output (upper half of the nodes) of the base load and the peaking power plant matches the corresponding expected residual load (lower half of the nodes). For instance, the base load power plant should provide 60 MW and the peaking power plant 20 MW in case of a residual load of 80 MW in t_1 (here, $t_i = t_{\text{now}} + i \cdot \Delta\tau$ with $i = \{1, 2, 3\}$). While the base load power plant should provide an output that is more or less constant and independent of changes in the residual load, the peaking power plant – whose maximum output is 20 MW – should make adjustments in accordance with residual load changes.

deal with the uncertainties captured in λ 's local deviation tree $\delta_\lambda^\mathcal{T}$ and, in turn, to meet the different demand scenarios in $D_\lambda^{\mathcal{T}, \text{dis}}$. That way, TBST-based schedules proactively guide the agents' decisions of how many resources to provide in which situation. An example illustrating this principle is depicted in Figure 11.3 where the combined scheduled output of a base load and a peaking power plant has to satisfy different residual load scenarios. In Chapter 12, we show how dispatchable agents use their schedule as a source of information for adequate reactive output adjustments that comply with the actual demand, and demonstrate that this approach improves the system's efficiency and robustness. To promote the dispatchable agents' ability to adapt their supply to unforeseen situations, intermediaries schedule additional reserves (i.e., they proactively increase the dispatchable agents' available degrees of freedom) as explained in Section 11.3.

The *online multi-stage stochastic optimization problem* (cf. [83]) the intermediary λ solves to create robust TBST-based schedules for its subordinate dispatchable agents \mathcal{V}_λ can be formalized as follows:

$$\underset{S_a^\mathcal{T}[n]}{\text{minimize}} \quad \alpha_\Delta \cdot \mathbb{E}(\Delta) + \alpha_\Gamma \cdot \mathbb{E}(\Gamma) \quad (11.1a)$$

$$\text{subject to} \quad \forall a \in \mathcal{V}_\lambda, \forall n \in D_\lambda^{\mathcal{T}, \text{dis}} \setminus \{r\} : \quad (11.1b)$$

$$\exists [x, y] \in L_a^{\theta(n)} : x \leq S_a^\mathcal{T}[n] \leq y, \quad (11.1c)$$

$$\vec{S}_a^{\min}(S_a^\mathcal{T}[f(n)]) \leq S_a^\mathcal{T}[n] \leq \vec{S}_a^{\max}(S_a^\mathcal{T}[f(n)]), \quad (11.1c)$$

$$\text{with} \quad \mathbb{E}(\Delta) = \sum_{n \in D_\lambda^{\mathcal{T}, \text{dis}} \setminus \{r\}} p(n) \cdot |S_{\mathcal{V}_\lambda}^\mathcal{T}[n] - D_\lambda^{\mathcal{T}, \text{dis}}[n]|, \quad (11.1d)$$

$$\mathbb{E}(\Gamma) = \sum_{\substack{a \in \mathcal{V}_\lambda, \\ n \in D_\lambda^{\mathcal{T}, \text{dis}} \setminus \{r\}}} p(n) \cdot \kappa_a(S_a^\mathcal{T}[n]), \quad (11.1e)$$

$$\text{and} \quad S_{\mathcal{V}_\lambda}^\mathcal{T}[n] = \sum_{a \in \mathcal{V}_\lambda} S_a^\mathcal{T}[n]$$

In this optimization problem, the scheduled contributions $S_a^\mathcal{T}[n]$ for the subordinate dispatchable agents $a \in \mathcal{V}_\lambda$ and nodes $n \in D_\lambda^{\mathcal{T}, \text{dis}} \setminus \{r\}$ are the decision variables. A dispatchable agent's current supply $S_a^\mathcal{T}[r]$ is represented by the root r of its schedule. Since scheduling is performed for the scheduling window \mathcal{W} , each node $n \in D_\lambda^{\mathcal{T}, \text{dis}}$ is assigned to a specific time step $t \in \mathcal{W} \cup \{t_{\text{now}}\}$ by means of the

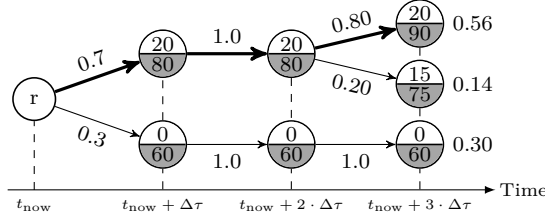


Figure 11.4: An example of a demand tree $D_{\lambda}^{\mathcal{T},dis}$ specifying the residual load the AVPP λ has to distribute (lower part of the nodes) and a corresponding schedule $S_a^{\mathcal{T}}$ for a peaking power plant $a \in \mathcal{V}_{\lambda}$ (upper part of the nodes): Here, we regard a scheduling window $\mathcal{W} = \{t_{\text{now}} + \Delta\tau, \dots, t_{\text{now}} + 3 \cdot \Delta\tau\}$. As for the demand tree $D_{\lambda}^{\mathcal{T},dis}$, each path from the root r to a leaf is a TBS that embodies a possible development of the residual load λ is accountable for. Except for r , every node represents an expected residual load in a future time step $t \in \mathcal{W}$ (all values in MW). With respect to the schedule $S_a^{\mathcal{T}}$, each node specifies the supply the peaking power plant should provide in case the actual residual load is equal to the corresponding expected residual load. For instance, if the actual residual load equals 80 MW in time step $t_{\text{now}} + 2 \cdot \Delta\tau$, the peaking power plant should provide an output of 20 MW. If the residual load is only 60 MW, the peaking power plant should be turned off. The values at the TBST's edges indicate the conditional probabilities that the residual load changes from one value to another. As stated in Section 5.2, we define the probability $p(n)$ that a node n occurs as follows: $p(r) = 1$, $n \neq r \rightarrow p(n) = p(n | f(n)) \cdot p(f(n))$, where the function f returns the parent of a node n . Hence, a TBS's probability of occurrence is equal to that of the corresponding leaf. The highlighted path indicates the *anticipated scenario*, i.e., the most likely development of the residual load.

function θ . We use this function to identify an agent's (possibly) time-dependent list of feasible supply ranges $L_a^{\theta(n)}$ (see Equation (11.1b)). In addition to $L_a^{\theta(n)}$, an agent's inertia is regarded by the functions \vec{S}_a^{\min} and \vec{S}_a^{\max} that restrict the agent's contribution $S_a^{\mathcal{T}}[n]$ for a node n depending on the agent's contribution $S_a^{\mathcal{T}}[f(n)]$ in n 's parent $f(n)$ (see Equation (11.1c)). To attain scalability by means of hierarchical system structures, the intermediary λ uses an abstracted control model for each subordinate intermediary $\lambda' \in \mathcal{I}_{\lambda}$ as described in Chapter 6. As before, λ 's goal is to find an allocation that meets the demand $D_{\lambda}^{\mathcal{T},dis}$ which it has to distribute as closely and cost-effectively as possible. Since each node $n \in D_{\lambda}^{\mathcal{T},dis}$ has a probability of occurrence $p(n)$, λ obtains a robust solution with regard to the primary objective by minimizing the *expected* total demand violation $\mathbb{E}(\Delta)$ (see Equation (11.1d)). This violation is defined as the expected value of the absolute deviation between the total scheduled contribution $S_{\mathcal{V}_{\lambda}}^{\mathcal{T}}[n]$ and the demand $D_{\lambda}^{\mathcal{T},dis}[n]$, calculated over all $n \in D_{\lambda}^{\mathcal{T},dis} \setminus \{r\}$ (by summing up *absolute* violations, negative and positive deviations cannot cancel each other out).³ Similarly, λ achieves a robust solution with regard to the secondary objective by minimizing the *expected* total costs $\mathbb{E}(\Gamma)$ that are based on the agent-specific cost functions κ_a (see Equation (11.1e)). Again, we use the parameters α_{Δ} and α_{Γ} to establish the desired prioritization of the two objectives (see Equation (11.1a)).

Note that the minimization of expected values \mathbb{E} complies with the general objective function proposed by stochastic programming.⁴ Based on the empirical discrete probability distributions provided by TBSTs, we thus obtain *robust solutions* [42] that allow the system to operate efficiently and effectively in various likely situations.

Figure 11.4 depicts an example of a TBST-based schedule $S_a^{\mathcal{T}}$ that is based on an exemplary demand

³Technically, we calculate the expected value of the demand violation over all time steps given the probabilities of the individual demand scenarios in the demand tree. It can be shown that our formulation based on nodes is equivalent to that, although $p(n)$ is not a probability distribution over all nodes.

⁴Optimizing expectations yields some kind of “soft robustness”. As opposed to “hard robustness”, “soft robustness” does not guarantee that each scenario's demand is met to a certain degree. Because this would require complete (and presumably exponential) search algorithms, obtaining “hard robustness” would prevent us from hierarchically decomposing the overall scheduling problem and from employing heuristics that make use of local knowledge, such as the auction-based scheduling mechanism TruCAOS.

tree $D_{\lambda}^{\mathcal{T},dis}$. By taking account of $D_{\lambda}^{\mathcal{T},dis}$'s structure and scheduling contributions for nodes (note that multiple TBSs might share a path from the root to a node within the TBST) depending on the scheduled contributions in the corresponding parent and child nodes, TBST-based schedules always represent feasible solutions. This means that we ensure that the transitions from one node to another comply with the regarded agent's control model. In Figure 11.4, for instance, the transitions from the schedule's root r to 20 MW and to 0 MW are guaranteed to be feasible. The same holds for the transitions from the scheduled output of 20 MW in $t_{\text{now}} + 2 \cdot \Delta\tau$ to an output of 20 MW or 0 MW in $t_{\text{now}} + 3 \cdot \Delta\tau$. That way, agents can delay their decision which contribution to make, i.e., according to which scenario to contribute, until the time step a branch in their schedule $S_a^{\mathcal{T}}$ has to be taken. By heading for the most promising direction (i.e., taking the most suitable branch), the system self-improves its ability to deal with uncertain demand.

Note – Dealing with Uncertain Supply by Means of TBSTs

Note that we do not use TBSTs to deal with uncertainties in the *dispatchable* agents' supply. If we did, we would be faced with the problem that changing the scheduled contribution of a dispatchable agent a_i in response to the TBST for a dispatchable agent a_j could cause a change in the TBST for a_i , whereupon the contribution of a_j would have to be adjusted and so on and so forth. It is not guaranteed that there is a fixed point, i.e., a steady state.

11.2 Top-Down Creation of Scenario-Based Schedules

In the TBST-based formulation of the scheduling problem, each intermediary $\lambda \in \mathcal{I}$ creates schedules for its subordinate dispatchable agents \mathcal{V}_{λ} on the basis of an individual demand tree $D_{\lambda}^{\mathcal{T},dis}$. In this section, we clarify how an intermediary's demand tree $D_{\lambda}^{\mathcal{T},dis}$ is actually defined. For the top-down creation of TBST-based schedules, we explain how the top-level intermediary Λ creates its own demand tree $D_{\Lambda}^{\mathcal{T},dis}$, and how the other intermediaries $\lambda' \in \mathcal{I} \setminus \{\Lambda\}$ derive their demand trees $D_{\lambda'}^{\mathcal{T},dis}$ from their schedules $S_{\lambda'}^{\mathcal{T}}$.

Before we explain the so-called *meso-level approach* – our intended method for the creation of robust TBST-based schedules –, we briefly sketch a much simpler and, at first glance, intuitive method, called *macro-level approach*. In our evaluation provided at the end of this section, we use an implementation of the macro-level approach to reveal the advantages of the more complex meso-level approach that allows intermediaries to anticipate and compensate for deviations locally, i.e., right where they occur.

Dealing with Uncertainties at the Macro-Level

In the *macro-level approach*, each non-top-level intermediary $\lambda \in \mathcal{I} \setminus \{\Lambda\}$ simply redistributes the demand stipulated in its schedule $S_{\lambda}^{\mathcal{T}}$ so that $D_{\lambda}^{\mathcal{T},dis} = S_{\lambda}^{\mathcal{T}}$. This approach clearly does not allow the intermediaries to take their local uncertainties in the form of their local deviation trees $\delta_{\lambda}^{\mathcal{T}}$ into account when creating schedules for their subordinates. Instead, uncertainties are exclusively regarded by the top-level intermediary at the macro level.

Here, the top-level intermediary Λ triggers the top-down creation of schedules by determining its local demand prediction $\hat{D}_{\Lambda}^{\mathcal{W}}$. To this end, it requests a demand prediction $\hat{D}_a^{\mathcal{W}}$ from each of its subordinate non-dispatchable agents \mathcal{U}_{Λ} and its subordinate intermediaries \mathcal{I}_{Λ} (demand predictions of subordinate intermediaries correspond to their own local demand predictions). The resulting local demand prediction $\hat{D}_{\Lambda}^{\mathcal{W}} = \sum_{a \in \mathcal{U}_{\Lambda} \cup \mathcal{I}_{\Lambda}} \hat{D}_a^{\mathcal{W}}$ is equivalent to the demand prediction $\hat{D}^{\mathcal{W}} = \sum_{a \in \mathcal{U}} \hat{D}_a^{\mathcal{W}}$ aggregated over all non-dispatchable agents. Here, the deviation tree $\delta_{\Lambda}^{\mathcal{T}}$ of Λ reflects the accuracy of previous aggregated demand predictions. By adding the prediction $\hat{D}_{\Lambda}^{\mathcal{W}}$ to each scenario in Λ 's deviation tree $\delta_{\Lambda}^{\mathcal{T}}$ (cf. the explanation provided in Section 5.2 and the example in Figure 5.2), Λ derives the demand tree $D_{\Lambda}^{\mathcal{T},dis}$ that specifies the demand it has to distribute to its subordinate dispatchable agents \mathcal{V}_{Λ} as described in Equation (11.1). As stated above, all non-top-level intermediaries $\lambda \in \mathcal{I} \setminus \{\Lambda\}$ understand

their schedules S_λ^π as the demand $D_\lambda^{\pi,dis}$ they have to redistribute to their subordinate dispatchable agents \mathcal{V}_λ . As for the schedule of the top-level intermediary Λ , we have $S_\Lambda^\pi = D_\Lambda^{\pi,dis}$.

The advantage of this approach is that it allows the top-level intermediary to detect statistical relationships between the non-dispatchable agents' behavior. However, considering uncertainties exclusively at the macro-level prevents other intermediaries from scheduling their subordinate dispatchable agents in a way that allows them to locally compensate for deviations between supply and demand that originate from their own subsystems. By contrast, an intermediary λ_1 might have to deal with demand scenarios that stem from deviations that usually occur in the subsystem of another intermediary λ_2 that is situated in a different subtree of the hierarchy. Consequently, deviations as well as corresponding countermeasures propagate through and possibly affect other parts of the system. This can turn out to be problematic, especially if the hierarchy evolved on the basis of an existing physical or organizational infrastructure. In power systems, the intermediaries λ_1, λ_2 might, for instance, reside at different voltage levels or in different subnetworks with severely limited power flows between them. But even in systems that do not impose such restrictions, an approach that anticipates and compensates for deviations locally can mitigate the effects of uncertainties more effectively than the macro-level approach. This is confirmed by the empirical evaluation provided at the end of this section.

Dealing with Uncertainties at the Meso-Level

The *meso-level approach* enables each subsystem to anticipate and compensate for deviations between supply and demand locally. Hence, it avoids the propagation of deviations to higher levels and other subtrees in the hierarchy. The main idea is to derive the demand tree $D_\lambda^{\pi,dis}$ of an intermediary $\lambda \in \mathcal{I}$ from its local deviation tree δ_λ^π . Consequently, $D_\lambda^{\pi,dis}$ captures the uncertainties in λ 's local demand predictions \hat{D}_λ^π and has the same shape as δ_λ^π . As described in Section 11.1, this allows λ to encode strategies that define how many resources its dispatchable agents \mathcal{V}_λ have to provide in which situation in order to locally balance supply and demand. If each intermediary fulfills this task, the utmost goal at the macro level – that is to satisfy the demand as accurately as possible – is achieved. Note that an intermediary's local deviation tree δ_λ^π intentionally does *not* capture uncertainties resulting from the local demand of subordinate intermediaries. This is not necessary because an intermediary that cannot compensate for its local uncertainties within its subsystem – and thus has to propagate at least a part of its local deviations to its superordinate intermediary – is indicative of an improper system structure. In such a situation, a reorganization has to be triggered that re-establishes a suitable distribution of uncertainties and degrees of freedom among the subsystems (see Part III). The self-organizing system structure further promotes the local compensation for deviations since it aims for subsystems that feature a similar ratio between uncertainty (i.e., predictability) and degrees of freedom in terms of controllability.

Again, the top-level intermediary Λ triggers the top-down creation of schedules by determining its local demand prediction \hat{D}_Λ^π . However, instead of reporting their own local demand prediction \hat{D}_λ^π to their superior, all non-top-level intermediaries $\lambda \in \mathcal{I} \setminus \{\Lambda\}$ now send the *anticipated local demand* \bar{D}_λ^π defined by the *anticipated scenario* of their local demand tree D_λ^π to their superordinate intermediary λ' . Hence, we now have $\hat{D}_{\lambda'}^\pi = \sum_{a \in \mathcal{U}_{\lambda'}} \hat{D}_a^\pi + \sum_{\lambda \in \mathcal{I}_{\lambda'}} \bar{D}_\lambda^\pi$ for the local demand prediction of an intermediary $\lambda' \in \mathcal{I}$. By reporting the *anticipated local demand* in lieu of the local demand prediction, intermediaries alleviate the effect of propagating uncertainties in the form of likely deviations to higher levels. By doing so, an intermediary λ affirms its superordinate intermediary that the anticipated local demand will occur. At the same time, this means that λ takes responsibility for scheduling its subordinate dispatchable agents \mathcal{V}_λ in a way that allows them to reactively and locally compensate for deviations from the anticipated local demand (strategies of how such reactions might look like are encoded in the subordinate dispatchable agents' schedules).

Example – Determining and Communicating the Anticipated Local Demand

The procedure of determining the anticipated local demand is depicted in the steps (1a) and (2) in

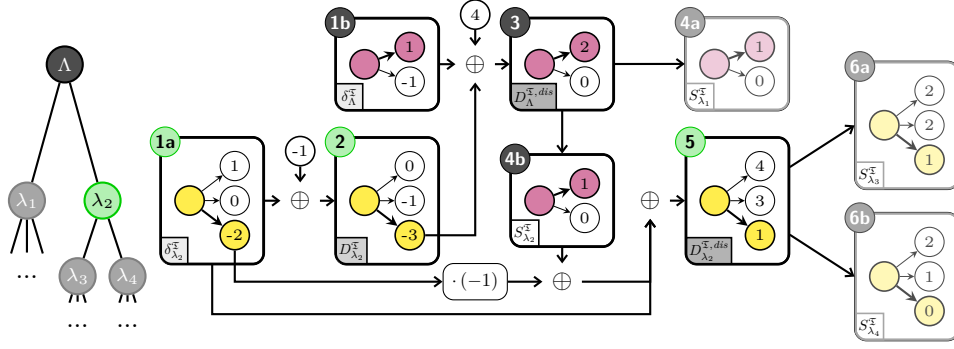


Figure 11.5: Top-down creation of TBST-based schedules according to the meso-level approach. The bold arcs in scenario trees highlight the anticipated scenario. The steps (1b), (3), (4a), and (4b) are performed by the top-level intermediary Λ , whereas the steps (1a), (2), (5), (6a), and (6b) are performed by the intermediary λ_2 . The value -1 between the steps (1a) and (2) represents λ_2 's local demand prediction. The local demand prediction of Λ is composed of λ_1 's anticipated local demand of 4 (shown between the steps (1b) and (3)) and λ_2 's anticipated local demand of -3 . The latter is extracted from λ_2 's local demand tree in step (2). The actions performed by the intermediaries λ_1 , λ_3 , and λ_4 are equivalent to those performed by λ_2 . For the sake of clarity, we do not depict these actions in the figure. We explain this figure step by step in the course of the examples given in this section.

Figure 11.5: The intermediary λ_2 combines its local deviation tree $\delta_{\lambda_2}^x$ (cf. step (1a)) with its local demand prediction of $\widehat{D}_{\lambda_2}^W = -1$ (note that λ_2 acquires its local demand prediction from its subordinate intermediaries λ_3 and λ_4 in the same manner). The combination yields λ_2 's local demand tree $D_{\lambda_2}^x$ (cf. step (2)). As for the anticipated scenarios, we have $\bar{\delta}_{\lambda_2}^x = -2$ for λ_2 's local deviation tree (i.e., the local demand is expected to be 2 less than predicted) and $\bar{D}_{\lambda_2}^x = -3 = -2 - 1$ for λ_2 's local demand tree. λ_2 sends its anticipated local demand $\bar{D}_{\lambda_2}^x = -3$ to its superordinate intermediary Λ . Further, we assume that the intermediary λ_1 sends the anticipated local demand $\bar{D}_{\lambda_1}^x = 4$ to Λ (cf. the value in the circle between the steps (1b) and (3)).

Having determined its local demand prediction \widehat{D}_{Λ}^W , the top-level intermediary Λ determines the demand $D_{\Lambda}^{x,dis}$ it has to distribute by adding \widehat{D}_{Λ}^W to its local deviation tree δ_{Λ}^x . For the creation of schedules for its subordinate dispatchable agents \mathcal{V}_{Λ} , Λ proceeds as explained in Section 11.1.⁵ Again, we have $S_{\Lambda}^x = D_{\Lambda}^{x,dis}$ for the schedule of the top-level intermediary Λ .

Example – Determining the Demand the Top-Level Intermediary Has to Distribute

The top-level intermediary Λ determines the demand $D_{\Lambda}^{x,dis}$ (cf. step (3) in Figure 11.5) it has to distribute by adding the local demand prediction $\widehat{D}_{\Lambda}^W = \bar{D}_{\lambda_1}^x + \bar{D}_{\lambda_2}^x = 4 - 3 = 1$ to each scenario in its local deviation tree δ_{Λ}^x (cf. step (1b)). Afterwards, it creates the schedules for its subordinate dispatchable agents λ_1 and λ_2 (cf. steps (4a) and (4b)). In the example, the schedules of λ_1 and λ_2 are

⁵ Each intermediary λ is responsible for compensating for deviations from the reported anticipated local demand. Due to the top-down creation of schedules, λ has to make sure that its superior λ' creates schedules that do not unnecessarily eat up λ 's degrees of freedom, i.e., its flexibility. That is because λ needs this flexibility in order to create schedules that allow for the local compensation for uncertainties in its local demand. These uncertainties are captured in λ 's local deviation tree. For this reason, λ slightly modifies its abstracted control model that is sent to λ' by adding information about the required flexibility, which is extracted from λ 's local deviation tree. λ' ensures that, if possible, the required flexibility is available.

identical. Both have to provide a supply of 1 if Λ 's local demand turns out to be 2. If Λ 's local demand equals 0, λ_1 and λ_2 should not make a contribution.

To redistribute the assigned fraction of the overall demand and, at the same time, incorporate uncertainties originating from its local demand, each intermediary $\lambda \in \mathcal{I} \setminus \{\Lambda\}$ extracts the *anticipated* scenario from its schedule S_λ^π , subtracts the *anticipated deviation* of its local deviation tree δ_λ^π and adds the resulting sequence to each scenario in δ_λ^π . This procedure yields λ 's demand tree $D_\lambda^{\pi,dis}$ specifying the demand it has to be distribute to its subordinate dispatchable agents \mathcal{V}_λ . Because $D_\lambda^{\pi,dis}$ is based on λ 's local deviation tree δ_λ^π , both trees have the same shape and $D_\lambda^{\pi,dis}$ captures the uncertainties in λ 's local demand predictions. Note that we have to subtract the anticipated scenario of δ_λ^π because this expected deviation was already included in the anticipated local demand \bar{D}_λ^π reported to λ 's superordinate intermediary. That means if the realization of λ 's local demand corresponds to the anticipated local demand \bar{D}_λ^π , λ just has to provide resources according to the anticipated scenario in its schedule S_λ^π . Because λ affirms its superordinate intermediary that the anticipated local demand \bar{D}_λ^π will occur, it creates schedules in a way that its subordinate dispatchable agents compensate for deviations from \bar{D}_λ^π .

Example – Determining the Demand a Non-Top-Level Intermediary Has to Distribute

The non-top-level intermediary λ_2 generates its demand tree $D_{\lambda_2}^\pi$ (cf. step (5) in Figure 11.5) by subtracting its anticipated local deviation $\bar{\delta}_{\lambda_2}^\pi = -2$ (cf. step (1a)) from the anticipated scenario $\bar{S}_{\lambda_2}^\pi = 1$ of its schedule (cf. step (4b)), and adding the result $\bar{S}_{\lambda_2}^\pi - \bar{\delta}_{\lambda_2}^\pi = 1 - (-2) = 3$ to all scenarios in its local deviation tree $\delta_{\lambda_2}^\pi$. λ_2 redistributes the demand $D_{\lambda_2}^\pi$ to its subordinate dispatchable agents λ_3 and λ_4 (cf. steps (6a) and (6b)). λ_3 and λ_4 use the same procedure to redistribute their fraction of the overall demand. Note that $D_{\lambda_2}^\pi$ and $\delta_{\lambda_2}^\pi$ have the same shape. Recall that $\bar{\delta}_{\lambda_2}^\pi = -2$ states that λ_2 's local demand is expected to be 2 less than predicted (i.e., -3 instead of -1). If the actual deviation turns out to be 0 or 1 instead of $\bar{\delta}_{\lambda_2}^\pi = -2$, λ_2 thus has to make a contribution of 3 or 4 instead of $\bar{S}_{\lambda_2}^\pi = 1$ in order to confront Λ with the (anticipated) local demand of $\bar{D}_\Lambda^\pi = -3$ and to satisfy the assigned demand of $\bar{S}_{\lambda_2}^\pi = 1$. For instance, in case of an actual deviation of $\delta = 1$, λ_2 has to make a contribution of $\bar{S}_{\lambda_2}^\pi + \delta - \bar{\delta}_{\lambda_2}^\pi = 1 + 1 - (-2) = 4$.

Note that an intermediary only combines the *anticipated scenario* \bar{S}_λ^π of its schedule with its local deviation tree δ_λ^π to obtain the demand tree $D_\lambda^{\pi,dis}$. We deliberately abstain from combining the complete schedule S_a^π with δ_λ^π because of the following reasons: The Cartesian product of S_a^π and δ_λ^π is not only likely to cause a huge number of scenarios at lower levels of the hierarchy, but also to yield TBSTs that do not accurately approximate the underlying stochastic process (i.e., they probably contain many unrealistic scenarios and inaccurate probabilities). Instead, it would be necessary to analyze correlations between the scenarios in S_a^π and δ_λ^π , and to join those that are related to each other. We leave such an investigation for future work. In our approach, intermediaries mitigate the effect of not considering the complete schedule S_a^π by scheduling each subordinate intermediary in a way that its utilization is, with regard to a specific time step, as independent of a specific TBS as possible. This is achieved by minimizing the mean absolute deviation of the scheduled contributions in each regarded time step in \mathcal{W} (deviations are weighted proportionally to the nodes' probabilities). As a result, local uncertainties are preferably addressed by engaging dispatchable agents that represent physical devices. Note that this procedure additionally reduces the propagation of uncertainties to lower levels. With regard to the power management case study, this means that AVPPs are scheduled like a specific type of base load power plant whose scheduled output for a specific time step is more or less independent of the different residual load scenarios.

In the end, each dispatchable agent $a \in \mathcal{V}$ has a schedule $S_a^{\mathcal{T}}$ for all scenarios of the local deviation tree $\delta_\lambda^{\mathcal{T}}$ of its superordinate intermediary λ . If each intermediary acts as described, the whole system achieves its primary goal of maintaining the balance between supply and demand.

Comparison of the Macro- and the Meso-Level Approach

In the following, we compare the macro- and the meso-level approach in our simulation environment for autonomous power systems. We primarily investigate their influence on scheduling times and the system's ability to anticipate inaccuracies in demand predictions. While we already identified the advantage of being able to switch to the most suitable demand scenario at runtime in Section 5.3, we want to reproduce these findings in the context of schedule creation at this place. For this purpose, we compare situations in which AVPPs schedule dispatchable power plants for the entire demand tree ("ma-TBST", "me-TBST") to others in which schedules are only created for the anticipated demand scenario ("ma-1-TBS", "me-1-TBS"). The prefixes "ma" and "me" stand for the macro- and the meso-level approach, respectively. Hereinafter, we refer to ma-TBST, me-TBST, ma-1-TBS, and me-1-TBS as "modes". The combination of a specific mode and system structure represents a certain evaluation scenario. For each of a total of 18 evaluation scenarios, we performed 100 simulation runs.

Test Bed

We base our evaluation on a system consisting of 173 dispatchable and 350 non-dispatchable power plants of different types (hydro, biofuel, gas power plants as well as solar plants and wind generators). As before, each power plant is modeled as an individual agent and non-dispatchable consumers are represented by a single agent that is assigned to the top-level AVPP. The demand, i.e., the residual load, originates from weather-dependent power plants and the consumers. We use real world data for the capabilities of physical power plants⁶ (such as minimal and maximal production boundaries), the load curves⁷, and the simulated weather conditions⁸ influencing the output of weather-dependent power plants. Each dispatchable power plant's inertia is defined within typical boundaries. The production costs, i.e., the unit price of electricity, range between $9.00 \frac{\text{euro cent}}{\text{kWh}}$ and $11.00 \frac{\text{euro cent}}{\text{kWh}}$.

The evaluation is implemented in a sequential, round-based execution model in which each round corresponds to a specific time step $t \in \mathcal{T}$. The dispatchable power plants have to satisfy a prescribed residual load over a period of 24 hours, corresponding to $|\mathcal{T}| = 96$ time steps, each representing a time span of 15 min. Every 15 min, AVPPs update their local deviation trees for the local residual load and create schedules for the next hour with a resolution of $\Delta\tau = 15 \text{ min}$ on the basis of residual load predictions provided by non-dispatchable power plants and consumers. Each schedule thus comprises $N = 4$ time steps. Dispatchable power plants provide resources according to the anticipated (local) residual load scenario. AVPPs employ the regio-central approach *RegioC* for schedule creation. The corresponding optimization problems are formulated as mixed integer linear programs and solved by the standard mathematical programming software IBM ILOG CPLEX⁹. With regard to Equation (11.1a), the parameters α_Δ and α_Γ are fixed in a way that establishes the desired prioritization among the different objectives.

Uncertainty in residual load predictions is generated by means of Markov models that randomly modify the actual behavior of non-dispatchable power plants and consumers. To reflect sequential dependencies in the predictions' accuracy (as is the case with the quality of weather predictions, for instance), the prediction error of a specific time step depends on prediction errors of previous time steps. The stochastic process is further designed in a way that short-term predictions are more precise than long-term predictions, and that predictions become more accurate as a future point in time approaches. The accuracy of predictions of non-dispatchable power plants within the same AVPP is coupled as is the

⁶Energymap (Bavaria), 2012: <http://www.energymap.info>, retrieved in 2012.

⁷LEW, 2012: <http://www.lew-verteilnetz.de>, retrieved in 2012.

⁸LfL (Bavaria), 2010: <http://www.lfl.bayern.de/agm/>, retrieved in 2012.

⁹IBM ILOG CPLEX Optimizer, Version 12.4, 2011: <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, retrieved on March 1, 2016.

	superflat	flat		deep		deeper		deepest	
		me	ma	me	ma	me	ma	me	ma
#Dispatchable Power Plants per AVPP	173.00	12.06		6.55		2.42		2.35	
TBST									
Max. Seq. Scheduling Time per Time Step [ms]	2060.27 (153.38)	413.12 (24.74)	420.34 (34.68)	313.67 (35.28)	297.24 (20.31)	336.31 (18.21)	303.40 (15.92)	408.32 (37.17)	368.66 (18.05)
Scheduling Time per AVPP and Time Step [ms]	2060.27 (153.38)	138.29 (22.79)	89.03 (14.48)	81.16 (3.29)	51.85 (5.16)	40.60 (1.26)	26.82 (0.38)	40.41 (1.08)	26.65 (0.32)
#TBSs per Schedule	2.55 (0.17)	2.26 (0.11)	1.29 (0.11)	2.28 (0.10)	1.26 (0.18)	2.22 (0.09)	1.10 (0.02)	2.20 (0.08)	1.12 (0.02)
Reduction of Prediction Error per AVPP and Time Step (Anticipated TBS) [%]	55.66 (3.46)	61.11 (2.18)	56.46 (2.98)	61.11 (1.95)	55.96 (3.61)	60.21 (1.93)	56.05 (3.35)	60.26 (1.84)	55.62 (3.31)
Reduction of Prediction Error per AVPP and Time Step (Best TBS) [%]	74.35 (1.55)	75.39 (0.93)	74.74 (1.53)	75.29 (1.01)	74.38 (1.54)	75.07 (0.95)	74.76 (1.53)	75.04 (0.90)	74.33 (1.67)
Unit Price of Electricity per AVPP and Time Step [euro cent/kWh]	9.37 (0.01)	9.63 (0.07)	9.64 (0.06)	9.64 (0.09)	9.63 (0.07)	9.65 (0.08)	9.66 (0.06)	9.65 (0.06)	9.66 (0.06)
1 TBS									
Max. Seq. Scheduling Time per Time Step [ms]	767.40 (6.86)	207.63 (8.95)	207.39 (8.41)	164.94 (4.20)	166.49 (3.66)	206.10 (9.04)	198.38 (6.18)	246.97 (7.74)	244.99 (7.57)
Scheduling Time per AVPP and Time Step [ms]	767.40 (6.86)	70.81 (10.31)	72.69 (10.54)	44.03 (0.79)	44.69 (0.73)	25.19 (0.27)	25.43 (0.30)	25.00 (0.24)	25.07 (0.23)
Unit Price of Electricity per AVPP and Time Step [euro cent/kWh]	9.36 (0.01)	9.63 (0.07)	9.46 (0.07)	9.62 (0.09)	9.47 (0.06)	9.66 (0.06)	9.49 (0.06)	9.65 (0.06)	9.50 (0.06)

Table 11.1: Evaluation results for me-TBST, ma-TBST, me-1-TBS, and ma-1-TBS in combination with different hierarchical structures of AVPPs. All results are averaged over 100 simulation runs for each evaluation scenario. Values in parentheses denote standard deviations. “Prediction error” stands for the difference between the expected residual load and the actual residual load. In case of me-1-TBS and ma-1-TBS, the results for the reduction of the prediction error correspond to the reduction by the anticipated TBS of the modes me-TBST and ma-TBST, respectively.

case with weather predictions for different locations. The top-level AVPP is able to measure the quality of residual load predictions with an accuracy of 1.5 % of the mean local residual load recorded in the experiences that are used for the generation of the local deviation tree. All other AVPPs can perceive erroneous predictions with an accuracy of 10.0 %. We use a higher resolution for the top-level AVPP because its local residual load includes the consumption and is thus much higher than the local residual load of the other AVPPs.

We performed our simulation runs on five different structures called “superflat”, “flat”, “deep”, “deeper”, and “deepest”. These structures are of height 1, 2, 3, 5, and 7, and exhibit an average number of 173, 12.06, 6.55, 2.42, and 2.35 dispatchable power plants per AVPP, respectively. They were generated offline on the basis of the same set of power plants. To examine the influence of these hierarchies on scheduling times and the dispatchable power plants’ ability to anticipate uncertainties in residual load predictions, we did not allow the agents to change the structures at runtime. Note that the macro- and the meso-level approach are equivalent when applied to a centralized system, i.e., the “superflat” structure, which is why we regard a total of 18 evaluation scenarios.

Results

Table 11.1 lists the results obtained for the 18 evaluation scenarios. For all modes, the structure “superflat” – which corresponds to a centralized scheduling approach – unsurprisingly yields the longest maximum sequential scheduling time (i.e., the longest of all serial paths that result from adding the scheduling times for each branch in the hierarchy; see Equation (6.1)). The structure “deep” represents a sweet spot where RegioC achieves the shortest max. seq. scheduling times for all modes. In case of me-TBST, for instance, RegioC is 84.78 % faster when using “deep” instead of “superflat”.

In mode ma-TBST, AVPPs consider only about 60 % of the TBSs regarded in mode me-TBST. That is because ma-TBST only allows the top-level intermediary to capture prediction errors. The lower number of TBSs in mode ma-TBST results from schedules in which AVPPs should provide the same output sequence in two or more scenarios. Consequently, two or more residual load scenarios considered by their superior can be regarded as a single scenario when they redistribute the assigned demand to subordinate dispatchable power plants. The fact that the number of TBSs considered in the “superflat” structure is even slightly larger than those for the other system structures in mode me-TBST confirms that we did not unfavorably handicap ma-TBST’s ability to anticipate prediction errors. In accordance with the difference in the number of TBSs, the mean scheduling time per AVPP is also about 60 % lower when using ma-TBST instead of me-TBST. ma-TBST’s max. seq. scheduling times are, however, only slightly shorter than those of me-TBST. This is indicative of a system structure in which the scheduling times of siblings are rather dissimilar, which most likely results from siblings that consider a different number of scenarios in mode ma-TBST. As for the unit price of electricity, there is no significant difference between ma-TBST and me-TBST, meaning that satisfying the residual load for a larger number of TBSs did not add to the costs.

While the top-level AVPP reduces the prediction error of the overall residual on the basis of the anticipated residual load scenario by an average of 56.02 % (standard deviation $\sigma = 0.30\%$) in case of ma-TBST, the prediction error is even reduced by 60.67 % ($\sigma = 0.44\%$) in case of me-TBST (averages are calculated over all structures exclusive of the “superflat” structure where ma and me are equivalent). Consequently, the larger number of TBSs the mode me-TBST takes into account can be regarded as a reasonable investment in the system’s robustness. However, as schedules are not only created for the anticipated residual load scenario but for a whole TBST, power plants can switch to the most suitable residual load scenario at runtime (see Chapter 12). In this case, the prediction error can even be reduced by an average of 74.57 % ($\sigma = 0.19\%$) in mode ma-TBST and by 75.20 % ($\sigma = 0.15\%$) in mode me-TBST. Note that me-TBST and ma-1-TBS yield a very similar reduction of the prediction error by means of the most suitable TBS, whereas me-TBST’s anticipated scenario obtains a significantly higher reduction of the prediction error than ma-TBST’s anticipated scenario. This demonstrates the advantage of enabling intermediaries to anticipate prediction errors locally (me-TBST) instead of dealing with uncertainties exclusively on the macro level (ma-TBST). Switching from one scenario to another is evidently not possible in case of ma-1-TBS and me-1-TBS, where schedules are only created for the anticipated residual load scenario.

While me-TBST and ma-TBST achieve average max. seq. scheduling times of 367.85 ms ($\sigma = 43.65$ ms) and 347.41 ms ($\sigma = 50.56$ ms), me-1-TBS and ma-1-TBS obtain an average of 206.41 ms ($\sigma = 1.96$ ms) and 204.31 ms ($\sigma = 1.80$ ms), respectively. The max. seq. scheduling times for a complete TBST is thus, on average, 1.5 as long as those for a single TBS, which is a sign of the complexity of multi-stage stochastic optimization problems. Nonetheless, the hierarchical system structure enables AVPPs to create TBST-based schedules in adequate time.

As for the unit price of electricity, Table 11.1 shows that the schedules created in the “superflat” structure are most cost-efficient, regardless of whether they are created for a complete TBST or only for a single TBS. As we could observe in our previous evaluations, runtime performance thus comes at the expense of an increased unit price of electricity. ma-1-TBS is slightly more expensive with average prices of $9.48 \frac{\text{euro cent}}{\text{kWh}}$ ($\sigma = 0.01 \frac{\text{euro cent}}{\text{kWh}}$). Even if the meso-level approach only takes account of a single TBS (cf. mode me-1-TBS), locally dealing with uncertainties increases costs to an average of $9.64 \frac{\text{euro cent}}{\text{kWh}}$ ($\sigma = 0.01 \frac{\text{euro cent}}{\text{kWh}}$). We ascribe this to the overhead resulting from the fact that positive and negative deviations cannot cancel each other out. The costs when creating schedules for a complete tree are very similar, $9.64 \frac{\text{euro cent}}{\text{kWh}}$ for me-TBST and $9.65 \frac{\text{euro cent}}{\text{kWh}}$ for ma-TBST (both $\sigma = 0.01 \frac{\text{euro cent}}{\text{kWh}}$). However, the meso-level approach allows AVPPs to reduce the influence of prediction errors right where they occur, thereby avoiding their propagation to other parts of the hierarchy.

To summarize, the evaluation confirms that the meso-level approach compensates for uncertainties more accurately than the macro-level approach. Furthermore, we could reproduce the advantage of being able to choose the most suitable scenario at runtime in the context of the schedule creation. Evidently, creating robust solutions to the scheduling problem comes at the expense of higher costs. However, as stated at the beginning of this chapter, approaches to solving optimization problems under

uncertainty generally presume that reducing the risk of not being able to deal with the actual demand outweighs higher costs. The hierarchical creation of TBST-based schedules enables the agents to come to a compromise between economic efficiency, runtime performance, and robustness.

11.3 Provision of Reserves in Hierarchical Systems

As motivated at the beginning of this chapter (see **(R1)**, **(R2)**, **(R3)**), it is crucial to enable dispatchable agents to deviate from their scheduled contributions in order to balance supply and demand. In Chapter 12, we discuss how the subordinate dispatchable agents $a \in \mathcal{V}_\lambda$ of an intermediary λ can use their schedules $S_a^\mathcal{T}$ as a blueprint for how to reactively adjust their supply to compensate for differences between λ 's actual and anticipated local demand. Although TBST-based schedules always contain feasible transitions (see Section 11.1), due to inertia, *spontaneous* switches (i.e., those after a branch had to be taken) from one targeted contribution to another are only possible if the dispatchable agents feature an adequate amount of reserves in the form of additional degrees of freedom. In a nutshell, the idea of provisioning reserves is to improve the system's ability to balance supply and demand by means of permanent reactive supply adjustments. Our self-organizing system structure promotes the provision of reserves by balancing the subsystems' ratio between uncertainty (i.e., predictability) and degrees of freedom in terms of controllability (see Part III).

For illustration, let us consider the schedule depicted in Figure 11.4 and assume that this schedule belongs to an AVPP consisting of base load as well as peaking power plants. Further, assume that the AVPP produces 5 MW in t_{now} and decided in t_{now} to deliver 20 MW in $t_{\text{now}} + \Delta\tau$ because the corresponding node has the highest conditional probability of 70 %. Consequently, the AVPP starts to increase its output in t_{now} so that it finally reaches its target of 20 MW in $t_{\text{now}} + \Delta\tau$. If, in a time step t between t_{now} and $t_{\text{now}} + \Delta\tau$, the AVPP realizes that the actual residual load in $t_{\text{now}} + \Delta\tau$ will be lower than expected (say 70 MW instead of 80 MW), it might have difficulties in decreasing its output fast enough (e.g., to corresponding 10 MW) if it ramped up slow base load power plants. This problem can even occur if it turns out that the residual load develops according to the lower scenario so that it reaches 60 MW in $t_{\text{now}} + \Delta\tau$. If the AVPP expected that it might have to decrease its output, it would be well-advised to increase the output of responsive peaking power plants that are able to decrease their output fast enough.

With regard to the scheduling problem, incorporating reserves, i.e., flexibility, is an additional objective that must not hinder an intermediary λ from satisfying its demand $D_\lambda^{\mathcal{T}, \text{dis}}$. Because the provision of reserves comes at a price, we propose that each intermediary λ uses its knowledge about prior reactive supply adjustments as feedback to estimate its locally *required* negative $R_\lambda^{\text{req}, -}[n]$ and positive $R_\lambda^{\text{req}, +}[n]$ reserves that should be available between two nodes $f(n), n \in D_\lambda^{\mathcal{T}, \text{dis}}$. This is why we speak of *feedback-driven* reserves. The feedback is provided by the algorithm dispatchable agents use to apply their schedules and to make reactive supply adjustments (see Chapter 12). When creating schedules, λ now not only distributes the demand $D_\lambda^{\mathcal{T}, \text{dis}}$ but also reserves to its subordinate dispatchable agents $a \in \mathcal{V}_\lambda$. In the following, we describe the scheduling of reserves from the perspective of an arbitrary intermediary λ , taken from somewhere in the hierarchy. We only refer to positive reserves in our explanations. The provision of negative reserves is analogously defined. Note that the expectations $\mathbb{E}(R^{\text{vio}})$ and $\mathbb{E}(R^{\text{vio}\downarrow})$ defined in Equations (11.2b) and (11.2c) refer to the expected *sum* of positive and negative reserve violations.

The reserves $R_\lambda^{\text{dis}, +}[n] = R_\lambda^{\text{req}, +}[n] + R_\lambda^{\text{ass}, +}[n]$ that λ has to *distribute* to its subordinate dispatchable agents \mathcal{V}_λ are defined by the sum of the locally *required* reserves $R_\lambda^{\text{req}, +}[n]$ and λ 's *assigned* reserves $R_\lambda^{\text{ass}, +}[n]$. The latter are prescribed in λ 's schedule $S_\lambda^\mathcal{T}$ that was created by λ 's superordinate intermediary (note that we have $R_\Lambda^{\text{ass}, +}[n] = 0$ for the top-level intermediary Λ). When scheduling reserves, λ tries to minimize the expected violation $\mathbb{E}(R^{\text{vio}})$ of the reserves $R_\lambda^{\text{dis}, +}[n]$ it has to distribute (see Equations (11.2b) and (11.2d)). λ 's ability to schedule reserves in a node $n \in D_\lambda^{\mathcal{T}, \text{dis}} \setminus \{r\}$ hinges on the *additional* reserves $R_a^{\text{add}, +}[n]$ λ 's subordinate agents $a \in \mathcal{V}_\lambda$ can promise to provide for λ in n (see Equation (11.2e)). The additional reserves $R_a^{\text{add}, +}[n]$ are subject to a 's *available* reserves $R_a^{\text{avl}, +}[n]$

as well as the reserves $R_a^{req\downarrow,+}[n]$ required by a 's subsystem. a 's available reserves $R_a^{avl,+}[n]$ mainly depend on a 's control model and the scheduled contribution in the nodes $f(n)$ and n (we provide an illustration of available positive reserves depending on an agent's scheduled supply at the end of this section). The reserves $R_a^{req\downarrow,+}[n]$ required by a 's subsystem are defined as follows: If a is an intermediary (i.e., if $a \in \mathcal{I}$), the reserves $R_a^{req\downarrow,+}[n] = R_a^{req,+}[n] + \sum_{\lambda' \in \mathcal{I}_a} R_{\lambda'}^{req\downarrow,+}[n]$ required by a 's subsystem are defined as the sum of a 's locally required reserves $R_a^{req,+}[n]$ and the sum of the reserves $R_{\lambda'}^{req\downarrow,+}[n]$ subordinate intermediaries $\lambda' \in \mathcal{I}_a$ require for their own subsystems. That way, a aggregates the locally required reserves of its entire subtree in the hierarchy. In this case, a 's additional reserves $R_a^{add,+}[n]$ are those reserves that are not needed to compensate for possible deviations in its own subtree of the hierarchy. The surplus can thus be provided to its superordinate intermediary. For instance, if a has $R_a^{avl,+}[n] = 12$ MW reserves available according to its current schedule, has locally required reserves of $R_a^{req,+}[n] = 1$ MW, and its subtree requires a total of $\sum_{\lambda' \in \mathcal{I}_a} R_{\lambda'}^{req\downarrow,+}[n] = 9$ MW reserves, it can provide its superior with $R_a^{add,+}[n] = 2$ MW additional reserves. Otherwise if a represents a physical device (i.e., if $a \notin \mathcal{I}$), it does not require any reserves for its subsystem so that $R_a^{req\downarrow,+}[n] = 0$ and $R_a^{add,+}[n] = R_a^{avl,+}[n]$.

To ensure that each subsystem represented by a subordinate intermediary $\lambda' \in \mathcal{I}_\lambda$ has the chance to schedule sufficient reserves itself, λ additionally tries to minimize the expected value $\mathbb{E}(R^{vio\downarrow})$ of the violations $R_{\lambda'}^{vio\downarrow,+}[n]$ of its subordinate subsystems' required reserves $R_{\lambda'}^{req\downarrow,+}[n]$ (see Equations (11.2c) and (11.2f)). To preserve the subsystems' autonomy and for a local compensation for deviations, we regard the minimization of $\mathbb{E}(R^{vio\downarrow})$ as more important than the minimization of $\mathbb{E}(R^{vio})$. Again, we use the parameters $\alpha_{R^{vio\downarrow}}$ and $\alpha_{R^{vio}}$ to obtain this prioritization (see Equation (11.2a)). While the satisfaction of the demand remains paramount, we degrade the minimization of the costs to reflect our risk-averse attitude. Hence, we extend the *objective function* of the scheduling problem formalized in Equation (11.1) as follows:

$$\underset{S_a^{\mathcal{T}}[n], R_a^{ass,+}[n]}{\text{minimize}} \quad \alpha_\Delta \cdot \mathbb{E}(\Delta) + \alpha_\Gamma \cdot \mathbb{E}(\Gamma) + \alpha_{R^{vio}} \cdot \mathbb{E}(R^{vio}) + \alpha_{R^{vio\downarrow}} \cdot \mathbb{E}(R^{vio\downarrow}) \quad (11.2a)$$

$$\text{with} \quad \mathbb{E}(R^{vio}) = \sum_{n \in D_\lambda^{\mathcal{T},dis} \setminus \{r\}} p(n) \cdot \left(R_\lambda^{vio,+}[n] + R_\lambda^{vio,-}[n] \right), \quad (11.2b)$$

$$\mathbb{E}(R^{vio\downarrow}) = \sum_{\substack{\lambda' \in \mathcal{I}_\lambda, \\ n \in D_\lambda^{\mathcal{T},dis} \setminus \{r\}}} p(n) \cdot \left(R_{\lambda'}^{vio\downarrow,+}[n] + R_{\lambda'}^{vio\downarrow,-}[n] \right), \quad (11.2c)$$

$$\text{and} \quad R_\lambda^{vio,+}[n] = \max \left\{ 0, R_\lambda^{dis,+}[n] - \sum_{a \in \mathcal{V}_\lambda} R_a^{add,+}[n] \right\}, \quad (11.2d)$$

$$R_a^{add,+}[n] = \max \left\{ 0, R_a^{avl,+}[n] - R_a^{req\downarrow,+}[n] \right\}, \quad (11.2e)$$

$$R_{\lambda'}^{vio\downarrow,+}[n] = \left| \min \left\{ 0, R_{\lambda'}^{avl,+}[n] - R_{\lambda'}^{req\downarrow,+}[n] \right\} \right|, \quad (11.2f)$$

Analogously for $R_\lambda^{vio,-}[n], R_a^{add,-}[n], R_{\lambda'}^{vio\downarrow,-}[n]$:

$$R_\lambda^{vio,-}[n] = \max \left\{ 0, R_\lambda^{dis,-}[n] - \sum_{a \in \mathcal{V}_\lambda} R_a^{add,-}[n] \right\},$$

$$R_a^{add,-}[n] = \max \left\{ 0, R_a^{avl,-}[n] - R_a^{req\downarrow,-}[n] \right\},$$

$$R_{\lambda'}^{vio\downarrow,-}[n] = \left| \min \left\{ 0, R_{\lambda'}^{avl,-}[n] - R_{\lambda'}^{req\downarrow,-}[n] \right\} \right|$$

Figure 11.6 illustrates the available positive reserves $R_a^{avl,+}[n]$ of a dispatchable agent $a \in \mathcal{V}$ between the two nodes $f(n)$ and n in its schedule $S_a^{\mathcal{T}}$. In general, the amount of available reserves is subject to the agent's control model and depend on whether the supply should be increased or decreased from $f(n)$ to n . Both cases are depicted in Figure 11.6. The corresponding cases for the available negative reserves can be obtained by reflecting the lines and points across the horizontal axes through the agent's

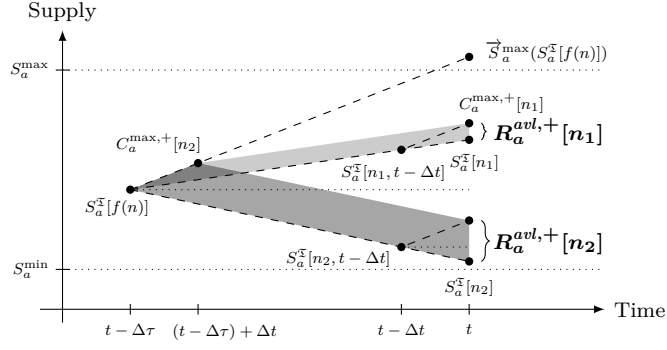


Figure 11.6: Two illustrations of the *available positive reserves* $R_a^{avl,+}[n]$ a dispatchable agent $a \in \mathcal{V}$ representing a physical device can provide between two nodes $f(n)$ and $n \in \{n_1, n_2\}$ (available reserves are indicated by the gray areas): n_1 stands for the case in which the agent a should increase its supply from $t - \Delta\tau$ to t , whereas n_2 represents the case in which a should decrease its supply. The amount of $R_a^{avl,+}[n]$ is subject to a 's scheduled supply $S_a^x[f(n)]$ and $S_a^x[n]$ as well as its control model. The latter captures a 's inertia represented by the function \vec{S}_a^{\max} as well as its minimal S_a^{\min} and maximal S_a^{\max} supply, among others. With regard to its scheduled supply $S_a^x[f(n)]$ and $S_a^x[n]$ (we assume a linear change of a 's supply between $f(n)$ and n), $R_a^{avl,+}[n]$ denotes, e.g., the *minimum* (in case of a *pessimistic* approach) of the maximal additional supply that a can mobilize within the single time steps between $f(n)$ and n . In this context, $C_a^{\max,+}[n]$ is the maximal supply a can provide between $f(n)$ and n if it increases its supply as much as possible from one time step to another. Clearly, $C_a^{\max,+}[n]$ is bounded by S_a^{\max} . If $S_a^x[f(n)] > S_a^x[n]$ (case $n = n_2$ in our illustration), a can provide more reserves than in case of $S_a^x[f(n)] < S_a^x[n]$ (case $n = n_1$ in our illustration) because it can also include reserves that result from not decreasing the supply from one time step to another.

supply $S_a^x[f(n)]$. Our calculations assume that reserves are to be mobilized from one time step to another, i.e., from t to $t + \Delta t$.

Note – Calculation of Available Positive Reserves

In accordance with Figure 11.6, the positive available reserves a dispatchable agent $a \in \mathcal{V}$ representing a physical device can provide from a time step t to a time step $t + \Delta t$ are calculated as follows:

$$\begin{aligned}
 R_a^{avl,+}[n] &= C_a^{\max,+}[n] - C_a^{ref,+}[n] + w_a[n] \\
 \text{with } C_a^{ref,+}[n] &= \max \{ S_a^x[f(n)], S_a^x[n] \}, \\
 C_a^{\max,+}[n] &= \min \{ S_a^{\max}, C_a^{ref,+}[n] + v_a[n] \}, \\
 v_a[n] &= (\vec{S}_a^{\max}(S_a^x[f(n)]) - C_a^{ref,+}[n]) / \Delta\tau \cdot \Delta t, \\
 w_a[n] &= \max \{ 0, S_a^x[f(n)] - S_a^x[n] \} / \Delta\tau \cdot \Delta t
 \end{aligned}$$

Here, $C_a^{ref,+}[n]$ corresponds to the maximum of the scheduled supply in $f(n)$ and n and serves as reference for calculating the available positive reserves. $C_a^{\max,+}[n]$ corresponds to the maximal possible supply that can be achieved between the nodes $f(n)$ and n . So the difference $C_a^{\max,+}[n] - C_a^{ref,+}[n]$ represents the available reserves the agent can provide by increasing its supply as much as possible from one time step to another (possibly faster than stipulated in its schedule). For situations in which the agent's schedule states that it should decrease its supply from $f(n)$ to n , $w_a[n]$ stands for positive reserves that stem from not decreasing the agent's supply.

Analogously, we obtain available negative reserves. When determining available reserves for an intermediary $\lambda \in \mathcal{I}$, $C_a^{\max,+}[n]$ is restricted to the intermediary's time-dependent list of feasible supply ranges L_λ^t instead of S_λ^{\max} and S_λ^{\min} .

Finally, an intermediary λ prescribes the assigned reserves $R_a^{ass,+}[n]$ for each subordinate dispatchable agent $a \in \mathcal{V}_\lambda$ proportionally to $R_\lambda^{dis,+}[n]$ and the agent's additional reserves $R_a^{add,+}[n]$, and inversely proportional to the subordinate agents' total additional reserves $R_{total}^{add,+}[n] = \sum_{a' \in \mathcal{V}_\lambda} R_{a'}^{add,+}[n]$ (with $R_\lambda^{ass,+}[n] = 0$):

$$R_a^{ass,+}[n] = \begin{cases} R_a^{add,+}[n] \cdot \min \left\{ 1, \frac{R_\lambda^{dis,+}[n]}{R_{total}^{add,+}[n]} \right\} & \text{if } R_{total}^{add,+}[n] \neq 0 \\ 0 & \text{else} \end{cases} \quad (11.3)$$

Note that the information about how many reserves may be assigned emerges in a bottom-up manner, whereas reserves are assigned in a top-down manner. The evaluation of alternative strategies for assigning reserves that could take additional optimization criteria into account, such as the dispatchable agents' costs, are subject to future work.

11.4 Robust and Cooperative Resource Allocation with TruCAOS

Our evaluation in Section 11.5 demonstrates that the decision to underpin the system's robustness by creating schedules for several demand scenarios, and the optimized provision of reserves complicate the search for high-quality solutions, especially in hierarchical systems consisting of relatively small subsystems. For this reason, we extend TruCAOS (see Chapter 10) – the auction-based heuristic for creating schedules in hierarchical systems – by the ability to obtain *robust solutions* by (1) creating schedules on the basis of demand trees and (2) optimizing the provision of required reserves in the form of additional degrees of freedom. In other words, we enable TruCAOS to solve the scheduling problem outlined in Sections 11.1 and 11.3. To this end, intermediaries deal with uncertainties in the local demand as described for the meso-level approach in Section 11.2.

In contrast to the regio-central approach in which intermediaries *assign* schedules to subordinates, TruCAOS enables intermediaries to distribute the demand in an iterative and incremental process that, in its basic form, is reminiscent of an iteratively performed first-price sealed-bid auction. By allowing an intermediary's dispatchable agents to sell or buy resources according to the demand that has to be distributed, they become an active part in the scheduling process. That way, TruCAOS does not accumulate the complexity of solving the scheduling problem in the intermediaries, which is advantageous to the creation of TBST-based schedules.

Only slight adjustments are necessary to obtain robust solutions with TruCAOS. As opposed to the basic procedure explained in Section 10.1, intermediaries now consider the demand that has to be distributed as a tree. Also the dispatchable agents' proposals and schedules have the shape of a tree. In all bidding iterations, an intermediary λ acting in the role of an auctioneer now has the goal to increase the *expected* satisfaction of the demand tree $D_\lambda^{\mathbb{E},dis}$. This corresponds to minimizing the expected total demand violation $\mathbb{E}(\Delta)$ in Equation (11.1a). In accordance with Equation (11.1d), the expected satisfaction is based on the demand scenarios' probabilities of occurrence.

Recall that, in TruCAOS, a schedule $S_a^{\mathbb{E}}$ does not become invalid if a new schedule creation is started. Instead, it is refined in the course of successive schedule creations. When initializing the *remaining demand* D_λ^{rem} that has to be distributed to subordinate dispatchable agents $a \in \mathcal{V}_\lambda$, λ therefore subtracts the agents' existing schedules from the initial demand tree $D_\lambda^{\mathbb{E},dis}$ that has to be distributed. However, due to dynamic uncertainties in λ 's local demand, the shape of $D_\lambda^{\mathbb{E},dis}$ can change from one schedule creation to another and thus differ from the shape of the existing schedules of λ 's subordinate dispatchable agents. For such situations, we use a greedy algorithm to find an adequate mapping of the scenarios of existing schedules to those in $D_\lambda^{\mathbb{E},dis}$. Such a mapping is determined by each intermediary before the first bidding iteration of a schedule creation. For nodes that were not regarded in the previous schedule creation (i.e., those outside the previous scheduling window), auctioneers and proposers assume a neutral contribution of zero. Because the resulting schedule might contradict an agent's control model, we reuse the dedicated phase in which the agents can correct their schedules. As described in Section 10.1, minimally inverse corrections are incentivized. This ensures that the agents do not misuse this feature for their own sake, which might impair the quality of the solution.

Having gathered the proposals of all bidders in a set \mathcal{P} , an intermediary completes the bidding iteration by identifying and accepting one or more suitable winner proposals $\mathcal{P}_w \subseteq \mathcal{P}$. For the selection of \mathcal{P}_w , we abide by the compromise between the number of bidding iterations and resulting monetary costs. To keep the costs down, intermediaries filter out, i.e., reject, proposals with a price-performance ratio worse than a historical average, calculated as a moving average over the last k_{ppr} schedule creations as explained in Section 10.2. As before, intermediaries select the winner proposals from the remaining set of valid proposals by solving the combinatorial auction problem. By allowing intermediaries to accept a combination of proposals, they can take advantage of synergy effects and reduce the number of bidding iterations. Besides being beneficial in multi-objective optimization, synergy effects are especially advantageous when creating TBST-based schedules because not only a single but several demand scenarios have to be satisfied.

Now, an intermediary λ chooses \mathcal{P}_w in such a way that there is no other combination that yields a greater *expected* gain in satisfaction of the remaining demand. In this context, λ 's secondary and tertiary goals are to minimize the expected violation $\mathbb{E}(R^{vio})$ of the reserves $R_\lambda^{dis,+}[n] = R_\lambda^{req,+}[n] + R_\lambda^{ass,+}[n]$ it has to distribute in the nodes $n \in D_\lambda^{T,dis} \setminus \{r\}$ (the same holds for negative reserves $R_\lambda^{dis-}[n]$) and to minimize the expected costs, respectively. Note that these objectives as well as their lexicographical order correspond to those defined in Equation (11.2a). The only but important difference is that λ does not have to care about the satisfaction of the required reserves on subordinate system levels by minimizing $\mathbb{E}(R^{vio\downarrow})$ (i.e., the expected violation of reserves required by subordinate subsystems) because each subordinate dispatchable agent $a \in \mathcal{V}_\lambda$ determines suitable contributions and additional reserves $R_a^{add,+}[n]$ it can provide for λ by itself. As λ is merely informed about $R_a^{add,+}[n]$ in the agents' proposals, computational costs are reduced. Analogously to Equation (11.3), TruCAOS assigns the reserves $R_a^{ass,+}[n]$ proportionally to $R_\lambda^{dis,+}[n]$ and $R_a^{add,+}[n]$, and inversely proportional to $R_{total}^{add,+}[n]$.

As before, each dispatchable agent $a \in \mathcal{V}$ creates its proposals by solving an optimization problem that corresponds to the one solved by intermediaries in the regio-central approach (see Equations (11.1) and (11.2)): The demand tree that has to be distributed corresponds to the demand specified in the *call for proposals* (CFP) the agent received from its intermediary. The set of agents schedules have to be created for only contains the agent a . While, in general, the dispatchable agents could incorporate individual and variable objectives when generating proposals, we assume that they generate their proposals as follows in order to remain competitive: According to the intermediaries' primary objective, a dispatchable agent $a \in \mathcal{V}$ creates proposals that maximize the expected satisfaction of the demand announced in the CFP. Maximizing the expected additional reserves $R_a^{add,+}[n]$ that can be provided to its intermediary λ is secondary. In case a is an intermediary, suggesting $R_a^{add,+}[n]$ makes sure that it has sufficient degrees of freedom to satisfy the reserves $R_a^{req\downarrow,+}[n]$ required by its subsystem when it is its turn to calculate schedules. This procedure leads to a top-down assignment of reserves. As before, we assume that agents stipulate their remunerations according to their cost functions κ_a (see Equation (11.1e)).

11.5 Evaluation in the Decentralized Power Management Case Study

The goal of our evaluation is to investigate if TruCAOS is more suitable for obtaining robust solutions than the regio-central approach *RegioC* explained in Section 2.2. For this purpose, we evaluate TruCAOS as well as RegioC in three different modes in the context of our case study: In the first mode, called "E-nu", demand predictions are not subject to uncertainties. In the second and third mode, called "E-nr" and "E-r", uncertainties are present but reserves are only scheduled in E-r. All modes were performed in combination with four different hierarchies of AVPPs, amounting to 12 evaluation scenarios. For each evaluation scenario, we performed 100 runs.

Test Bed

The basic setting of the evaluation environment corresponds to the one sketched in the evaluation provided at the end of Section 11.2. Here, the production costs, i.e., the average costs of providing a

contribution, range from $6.50 \frac{\text{euro cent}}{\text{kWh}}$ to $17.50 \frac{\text{euro cent}}{\text{kWh}}$. Power plants have to satisfy a prescribed residual load over a period of half a day, corresponding to 240 time steps, each representing $\Delta t = 3$ min. Every 15 min, AVPPs update their local deviation trees and create schedules for the next hour with a resolution of $\Delta \tau = 15$ min on the basis of residual load predictions. Again, schedules comprise $N = 4$ time steps. Given that schedules are recalculated every 15 min, required reserves only have to be scheduled for the next 15 min, i.e., for all TBST nodes $n \in D_{\lambda}^{\mathcal{T}, \text{dis}}$ that map to a time step $t \in \mathcal{W}$ with $t - t_{\text{now}} = 15$ min. Each AVPP $\lambda \in \mathcal{I}$ uses the maximum of its negative and positive reactive supply adjustments within the last 30 min as required negative $R_{\lambda}^{\text{req}, -}[n]$ and positive $R_{\lambda}^{\text{req}, +}[n]$ reserves. Reactive supply adjustments are made using the procedure explained in Chapter 12.

The optimization problems that have to be solved to generate proposals and determine winner proposals in TruCAOS as well as those of RegioC are formulated as mixed integer linear programs and solved by IBM ILOG CPLEX. Regarding Equation (11.2), the parameters α_{Δ} , α_{Γ} , $\alpha_{R^{\text{vio}}\downarrow}$, and $\alpha_{R^{\text{vio}}}$ are fixed in a way that establishes the desired prioritization among the different objectives. For TruCAOS, we initialize the parameters introduced in Sections 10.1 and 11.4 as follows: AVPPs consider the remaining residual load satisfied in case its absolute values are below $D_{\max}^{\text{rem}} = 5$ kW. As for the incremental distribution of the remaining residual load D_{λ}^{rem} , AVPPs use a fraction $g = 0.2$ of D_{λ}^{rem} if $D_{\lambda}^{\text{rem}} \cdot g > 1000$ kW, and $g = 1$ otherwise. The average price-performance ratio used to filter out inappropriate proposals is calculated over the last $k_{\text{ppr}} = 5$ schedule creations.

To keep the scheduling times of RegioC within reasonable bounds, each AVPP stops the schedule creation after 15 s if a feasible solution is present. If not, it gives CPLEX another 10 min. If this threshold is exceeded, no solution is found and the run is aborted. In most cases, the time span of 15 s is sufficient to create high-quality schedules.

We examine the following four questions of interest: (1) *Does TruCAOS outperform RegioC in terms of runtime performance?* (2) *Does TruCAOS obtain better results w.r.t. expected demand satisfaction, provision of required reserves, and costs?* (3) *How does the hierarchical system structure affect solution quality?* (4) *Are hierarchical problem decomposition and heuristics, such as TruCAOS, adequate means to deal with the complexity of solving multi-stage stochastic optimization problems and scheduling reserves?*

To investigate the impact of hierarchical system structures on TruCAOS's and RegioC's performance, we performed our evaluations on four different structures called “superflat”, “flat”, “deep”, and “deeper” of height 1, 2, 3, and 5, and with an average number of 173, 14.23, 6.44, and 2.42 dispatchable power plants per AVPP, respectively. All these structures were generated offline on the basis of the same set of power plants. To be able to analyze the hierarchies' influence on the algorithms' performance, we disabled the system's ability to change the structures at runtime.

Results

Table 11.2 shows the results for the three modes E-nu, E-nr, and E-r in combination with the four different system structures. The characteristics of the results for mode E-nu – in which residual load predictions are not subject to uncertainties – are similar to those we obtained for different hierarchical system structures in Section 10.5. Differences originate from the prescribed residual load, the evaluated time span, and the changes made to TruCAOS in Section 11.4.

In mode E-nu, RegioC achieves its shortest maximum sequential scheduling time (i.e., the longest of all serial paths that result from adding the scheduling times for each branch in the hierarchy; see Equation (6.1)) of 118.08 ms with structure “deep”. However, these are accompanied by significant violations of the assigned residual load of 7453.83 kW and relatively high mean total costs of the schedules of 164.89 k€ per time step. Note that we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$, that is, the maximum unit price of electricity, for violated assigned residual loads. Hence, the algorithms could not hold down costs by scheduling less electricity than required. Based on the “superflat” structure, RegioC obtains its lowest violation of the assigned residual load of only 0.10 kW and its lowest costs of 161.91 k€. The max. seq. scheduling time increases to 5652.60 ms, though. These observations are in line with those we made in the evaluation of the self-organized hierarchical problem decomposition (see Section 9.2). In case of TruCAOS, “superflat” again turns out to be the best structure for the considered set of power plants: The max. seq. scheduling time of 330.37 ms is remarkably short, especially given that the violation of the

E-nu	superflat		flat		deep		deeper	
	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS
#Runs Exceeding the Max. Scheduling Time	0	—	0	—	0	—	0	—
Max. Seq. Scheduling Time per Time Step [ms]	5652.60 (1022.87)	330.37 (69.73)	150.86 (34.45)	514.88 (265.77)	118.08 (34.05)	948.46 (122.84)	124.73 (33.23)	1684.54 (122.20)
Scheduling Time per AVPP and Time Step [ms]	5652.60 (1022.87)	330.37 (69.73)	60.85 (20.42)	173.49 (107.96)	31.56 (12.21)	174.07 (91.13)	16.91 (6.64)	100.78 (99.17)
Total Violation of the Assigned Residual Load per Time Step [kW]	0.10 (0.02)	0.35 (0.75)	225.31 (884.35)	67.12 (309.12)	7453.83 (2944.87)	387.16 (1031.02)	42875.82 (6706.31)	9670.69 (9525.34)
Total Costs of Schedules per Time Step [1000 €]	161.91 (24.41)	156.19 (24.64)	163.82 (24.24)	159.32 (20.93)	164.89 (21.27)	153.56 (23.29)	173.99 (21.33)	169.51 (16.85)

E-nr	superflat		flat		deep		deeper	
	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS
#Runs Exceeding the Max. Scheduling Time	39	—	0	—	0	—	0	—
#TBSs per Schedule	1.62 (0.96)	1.75 (1.15)	1.98 (1.30)	1.95 (1.29)	1.95 (1.32)	1.92 (1.30)	1.83 (1.32)	1.83 (1.32)
Max. Seq. Scheduling Time per Time Step [s]	46.37 (77.51)	1.10 (0.79)	0.43 (0.34)	1.23 (0.85)	0.22 (0.10)	1.87 (0.55)	0.20 (0.07)	2.98 (0.65)
Scheduling Time per AVPP and Time Step [s]	46.37 (77.51)	1.10 (0.79)	0.14 (0.17)	0.34 (0.31)	0.06 (0.04)	0.28 (0.20)	0.03 (0.02)	0.15 (0.16)
Total Exp. Violation of the Assigned Residual Load per Time Step [kW]	10.92 (127.28)	0.54 (0.98)	457.95 (1066.28)	156.77 (477.06)	8212.31 (5100.85)	1123.09 (1637.33)	39572.63 (13010.64)	13756.14 (9831.97)
Total Exp. Costs of Schedules per Time Step [1000 €]	150.67 (38.45)	153.85 (38.69)	163.44 (33.83)	167.47 (32.31)	165.50 (32.45)	162.66 (35.98)	177.16 (28.80)	177.41 (26.33)
Total Exp. Violation of Req. Reserves per Time Step [kW]	41679.20 (2101.53)	41556.09 (2595.41)	64017.60 (3298.07)	74053.92 (4961.36)	66985.54 (3371.81)	70029.36 (2957.42)	73376.43 (3929.13)	87332.14 (9571.86)

E-r	superflat		flat		deep		deeper	
	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS	RegioC	TruCAOS
#Runs Exceeding the Max. Scheduling Time	68	—	20	—	7	—	5	—
#TBSs per Schedule	1.65 (0.97)	1.76 (1.18)	1.97 (1.29)	1.96 (1.31)	1.97 (1.34)	1.96 (1.36)	1.85 (1.34)	1.82 (1.31)
Max. Seq. Scheduling Time per Time Step [s]	126.38 (129.21)	1.45 (1.09)	111.80 (213.37)	1.64 (0.91)	27.94 (59.84)	2.66 (0.78)	16.68 (20.15)	4.61 (1.18)
Scheduling Time per AVPP and Time Step [s]	126.38 (129.21)	1.45 (1.09)	10.20 (64.49)	0.42 (0.38)	1.51 (11.37)	0.36 (0.30)	0.25 (2.11)	0.20 (0.25)
Total Exp. Violation of the Assigned Residual Load per Time Step [kW]	60.04 (1464.10)	0.45 (0.93)	1131.58 (11309.59)	164.28 (516.05)	13250.51 (13672.30)	1089.03 (1704.78)	52904.41 (20112.41)	12526.38 (7893.77)
Total Exp. Costs of Schedules per Time Step [1000 €]	149.69 (38.09)	153.92 (38.54)	161.67 (34.04)	167.06 (32.39)	166.92 (31.95)	163.38 (36.03)	175.29 (29.36)	177.73 (26.09)
Total Exp. Violation of Req. Reserves per Time Step [kW]	9.93 (24.63)	1.50 (14.96)	15805.11 (2417.76)	8678.90 (1637.53)	27095.42 (2667.58)	7886.57 (1352.67)	42192.57 (3734.92)	20106.47 (2686.00)

Table 11.2: Evaluation results obtained by TruCAOS and RegioC for modes E-nu (no uncertainties, optimized provision of reserves disabled), E-nr (uncertain residual load, optimized provision of reserves disabled), and E-r (uncertain residual load, optimized provision of reserves enabled) in combination with different hierarchies. “#Runs Exceeding the Max. Scheduling Time” denotes the number of RegioC runs that were aborted due to an exceeded maximal scheduling time of 10 min. All other results are averaged over the 100 runs per evaluation scenario. Values in parentheses denote standard deviations. Apart from mode E-nu, all scheduling times are presented in seconds. To avoid that an AVPP can decrease its costs by not covering its assigned residual load, we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$ for violated assigned residual loads.

assigned residual load only amounts to 0.35 kW. TruCAOS’s costs of 156.19 k € are even 3.53 % below those of RegioC. These numbers highlight the benefit of enabling all dispatchable agents to actively participate in the process of schedule creation.

Still regarding mode E-nu, TruCAOS achieves much lower violations of the assigned residual load than RegioC in regard to deep hierarchies consisting of small AVPPs. In case of the structure “deep”, for instance, TruCAOS’s residual load violation is only 5.19 % of RegioC’s. The principle of schedule corrections that allows TruCAOS to create high-quality schedules in fragmented systems comes at the

expense of longer scheduling times, though. This is why TruCAOS's max. seq. scheduling time amounts to an average of 948.46 ms in the "deep" hierarchy. The same applies to the other hierarchies. Apart from that, we explain TruCAOS's increase of the max. seq. scheduling time with the height of the hierarchy by the overhead introduced by additional hierarchy levels (cf. Section 10.5): First, due to the complexity of the AVPPs' control models, AVPPs need more time for generating proposals than physical power plants. Second, additional hierarchy layers introduce additional overhead because more AVPPs have to solve the combinatorial auction problem, and, depending on their relative position in the hierarchy, these problems have to be solved sequentially, thereby adding to the max. seq. scheduling time. Top-down dependencies also restrict the parallel creation of proposed schedules to specific subsets of dispatchable power plants.

When disabling TruCAOS's price-performance filtering discussed in Sections 10.2 and 11.4, we obtain an algorithm that, with regard to the costs, randomly assigns schedules to power plants. In this case, the mean costs over all hierarchies are 171.80 k € (standard deviation $\sigma = 2.25$ k €) in mode E-nu. Given that TruCAOS attains mean costs of 159.65 k € ($\sigma = 6.05$ k €), this demonstrates the importance of optimizing schedules with regard to the resulting costs. RegioC yields mean costs of 166.15 k € ($\sigma = 4.65$ k €) that are lower than the costs of the random approach, but notably higher than those of TruCAOS.

When creating schedules in the presence of uncertain residual load predictions in modes E-nr and E-r, we observe that – with regard to a specific combination of system structure and mode – TruCAOS and RegioC had to deal with quite the same degree of uncertainties. This is mirrored in the mean number of considered TBSs in each schedule creation. Variations between different system structures result from different local residual loads.

The computational complexity introduced by the creation of TBST-based schedules and the optimized provision of reserves is reflected in the max. seq. scheduling times: TruCAOS's mean max. seq. scheduling time over all structures increases from 0.87 s ($\sigma = 0.52$ s) for E-nu, over 1.80 s ($\sigma = 0.74$ s) for E-nr, to 2.59 s ($\sigma = 1.26$ s) for E-r. As for RegioC, it raises from 1.51 s ($\sigma = 2.39$ s) for E-nu, over 11.81 s ($\sigma = 19.96$ s) for E-nr, to 70.70 s ($\sigma = 48.83$ s) for E-r. Note that the growth of the max. seq. scheduling time is much lower in case of TruCAOS than in case of RegioC. Above that, the mean costs over all hierarchies obtained by the random approach increase from 171.80 k € ($\sigma = 2.25$ k €) for E-nu, over 179.66 k € ($\sigma = 2.89$ k €) for E-nr, to 183.05 k € ($\sigma = 1.90$ k €) for E-r. For TruCAOS, we also observe an increase from 159.65 k € ($\sigma = 6.05$ k €) for E-nu, over 165.35 k € ($\sigma = 8.51$ k €) for E-nr, to 165.52 k € ($\sigma = 8.52$ k €) for E-r. Interestingly, the mean costs actually decline in case of RegioC: from 166.15 k € ($\sigma = 4.65$ k €) for E-nu, over 164.19 k € ($\sigma = 9.40$ k €) for E-nr, to 163.39 k € ($\sigma = 9.28$ k €) for E-r. When taking a closer look at the different evaluation scenarios, we realize that this decrease mainly stems from the "superflat" structure, as the costs of the other structures remain relatively stable. We ascribe this observation to the increase in the optimization problem's complexity. This counterintuitive explanation results from the fact that CPLEX comes up with feasible but not necessarily optimal solution after 15 s in mode E-nu. In case of the more complex optimization problems E-nr and E-r, CPLEX is occasionally not able to find a solution within 15 s so that it has to rely on the extra 10 min. This extra time often leaves RegioC enough room to create more cost-efficient schedules than in mode E-nu.

With regard to RegioC and E-r, we notice that RegioC's average max. seq. scheduling time significantly declines with the height of the hierarchy; from 126.38 s to 16.68 s. This comes along with a decrease in the average scheduling time per AVPP from 126.38 s to 0.25 s as well as a remarkably drop in the number of runs without a solution, i.e., those that had to be aborted by RegioC due to an exceeded maximal scheduling time of 10 min. The expected violation of the assigned residual load increases with the height of the hierarchy from 60.04 kW to 52.90 MW (the mean residual load was about 1.5 GW), though. The same applies to the total costs of the schedules per time step which rise by about 25 k € from "superflat" to "deeper", and the violation of required reserves. All this can be attributed to a higher degree of the system's fragmentation and errors introduced by model abstraction. While "deeper" has the least number of aborted runs, "deep" yields an acceptable trade-off between solution quality, scheduling times, and aborted runs for E-r. The same observations can be made for E-nr, where "flat" yields an acceptable trade-off. While these observations highlight the benefit of problem decomposition, they also demonstrate the necessity of the techniques presented in Part III that allow the system to come to such compromises in a self-organized manner. Due to the optimized provision of reserves in E-r,

RegioC's violation of reserves is considerably smaller in E-r than in E-nr. However, compared to E-nr, RegioC's number of aborted runs is significantly higher for E-r. Moreover, RegioC obtains a shorter average max. seq. scheduling time, which is between 46.37 s and 0.20 s, and a lower average expected violation of the assigned residual load, ranging between 10.92 kW and 39.57 MW in mode E-nr.

With regard to TruCAOS, we also notice that hierarchical problem decomposition leads to a decline in the average scheduling time per AVPP; from 1.10 s to 0.15 s in case of E-nr and from 1.45 s to 0.20 s in case of E-r. In contrast to RegioC, TruCAOS's max. seq. scheduling time increases with the height of the hierarchy, though (cf. the discussion of the results for mode E-nu). In spite of this behavior, TruCAOS does not only achieve much lower max. seq. scheduling times for E-r (1.45 s for "superflat") than RegioC (27.94 s for "deep"), but it also obtains better solutions in terms of the violation of the assigned residual load. While fragmentation also leads to a growth of the expected violation of the assigned residual load with the hierarchy's height, it only increases from 0.54 kW to 13.76 MW in case of E-nr and from 0.45 kW to 12.53 MW in case of E-r. In particular, the values for "superflat" and "flat" are remarkably low. Compared to RegioC, TruCAOS's accuracy in case of E-nr comes at the expense of higher max. seq. scheduling times though: 1.10 s for "superflat" and TruCAOS, compared to 0.43 s in case of "flat" and RegioC. Further, TruCAOS is able to satisfy the required reserves in E-r's "superflat" runs very accurately (average violation of 1.50 kW); also note that TruCAOS completed all runs as opposed to RegioC. Compared to RegioC, the average violation of reserves of 7.89 MW in case of "deep" – which constitutes RegioC's sweet spot for E-r where it achieves an average violation of 27.10 MW – is also very low. Given that RegioC's average expected costs range between approximately 150 k € and 177 k €, TruCAOS achieves very promising results of about 154 k € for "superflat". The benefit of TruCAOS's price-performance filtering becomes again evident when disabling its functionality: Then, the costs rise to approximately 180 k € in case of "superflat" and E-r.

We observe that incorporating reserves into the scheduling problem drastically decreases the violation of the required reserves. Note that an AVPP that cannot provide its required reserves might have to ask other AVPPs for reactive supply adjustments. However, these AVPPs did not expect this request when they created the schedules for their subordinates. Therefore, it is very likely that these supply adjustments come at unnecessarily high costs. Situations in which AVPPs repeatedly have to ask their neighbors for reactive supply adjustments are indicative of an improper system structure. A reorganization that, in terms of homogeneous partitioning, balances the AVPPs' ratio between uncertainties and controllability re-establishes the system's ability to locally deal with uncertainties (see Part III). That way, dependencies on the scheduled reserves of other AVPPs are mitigated. This results in loosely coupled subsystems, which is beneficial with regard to the decomposition of the scheduling problem (see Sections 6.3 and 9.2).

In an additional evaluation setting, we obtained results that confirm that TruCAOS's runtime performance actually profits from hierarchical problem decomposition under certain circumstances, as is the case with RegioC. Basically, this setting corresponds to the mode E-nr, but it complicates the search for high-quality robust solutions by exposing the system to a higher degree of uncertainty, which is reflected in a larger average number of scenarios per schedule of 2.76 ($\sigma = 0.25$). As a result, TruCAOS reaches its lowest max. seq. scheduling times of 1.37 s on average ($\sigma = 0.80$ s) using the "flat" structure. The average max. seq. scheduling times for the other system structures are 1.97 s ($\sigma = 1.07$ s), 2.20 s ($\sigma = 0.59$ s), and 3.58 s ($\sigma = 0.89$ s) for the structures "superflat", "deep", and "deeper", respectively.

To sum up, we observed that hierarchical problem decomposition and heuristics, such as TruCAOS, are appropriate means to deal with the complexity of solving multi-stage stochastic optimization problems and scheduling reserves. As adequate structures that achieve a suitable trade-off between solution quality and scheduling times depend on many factors, such as the size of the system, the regarded optimization problem, and the uncertainties involved, they have to be formed at runtime using the techniques introduced in Part III. Furthermore, our empirical evaluation confirms that TruCAOS significantly outperforms RegioC in terms of solution quality and runtime performance, in particular, when creating schedules in uncertain environments. TruCAOS is not only able to deal with larger subsystems but even takes advantage of their combinatorial variety. In turn, this promotes its ability to find high-quality robust solutions to the scheduling problem.

Chapter Summary and Outlook

In this chapter, we explained how dispatchable agents obtain robust solutions to the scheduling problem in the light of volatile demand that follows different behavioral patterns. To allow the agents to deal with different demand scenarios and, at the same time, lay the foundations for reactive supply adjustments, we combined techniques from online stochastic optimization with the provision of feedback-driven reserves. In this context, intermediaries use Trust-Based Scenario Trees (TBSTs) to derive empirical discrete probability distributions of prediction errors. To cope with systems of large size, we formalized the TBST-based scheduling problem for self-organizing hierarchical system structures. Further, we extended the auction-based scheduling mechanism TruCAOS for the creation of TBST-based schedules and the provision of reserves. To avoid that deviations between supply and demand propagate through the system, we explained how to create schedules that allow an intermediary and its subordinate agents to anticipate and compensate for inaccurate demand predictions locally, i.e., in their subsystem. Our empirical evaluation demonstrates the advantage of this approach. Especially when searching for robust solutions to the scheduling problem, TruCAOS clearly outperforms the regio-central approach in terms of scheduling times, demand satisfaction, provision of reserves, and costs.

In future work, we will investigate if the system's robustness further increases when merging an intermediary's demand tree with multiple scenarios of its schedule (see Section 11.2). In order to derive a meaningful TBST in this situation, the intermediary has to identify correlations between the prediction errors of its own local demand and those of the demand at higher system levels. As for TruCAOS, we want to enable heterogeneous agents, e.g., peaking and base load power plants, to follow individual preferences and learn strategies for making proposals that match their type, thereby increasing the overall system's performance. For instance, overall costs could be decreased if base load power plants proposed schedules in which their output is not subject to the different residual load scenarios. Moreover, we will examine possibilities for obtaining a fair allocation of resources. As fairness increases the agents' willingness to participate, it is a necessary property en route to the practical application of a scheduling algorithm like TruCAOS. Given that schedules address multiple demand scenarios and also incorporate the provision of reserves, appropriate fairness metrics have to be identified that take account of the different responsibilities an agent can take on in the system.

In the following chapter, we show how dispatchable agents meet the actual demand by reactively adapting their supply in compliance with their schedules. In this context, dispatchable agents utilize their TBST-based schedules as a blueprint for how much resources to provide in which situation.

Proactively Guided Reactive Supply Adjustments

Summary. Due to the dispatchable agents' inertia, their supply has to be specified proactively by the means of schedules. The scheduling problem's complexity and the uncertainties involved, however, call for rather coarse-grained schedules although the demand has to be satisfied for a much finer time pattern. Furthermore, deviations between supply and demand stemming from inaccurate demand predictions and from dispatchable agents that cannot comply with their schedules have to be cushioned. To compensate for deviations between the actual demand and supply, the dispatchable agents thus have to make reactive adjustments. These, however, have to be made in compliance with their scheduled contribution to be able to satisfy the future demand despite the agents' inertia, and to respect the objectives that have been regarded during the schedules' creation. In this chapter, we show that schedules based on *Trust-Based Scenario Trees* (TBSTs) proactively guide the dispatchable agents' permanent reactive supply adjustments. Our evaluation confirms that taking account of the information provided by TBST-based schedules allow the dispatchable agents to respect those objectives that can be encoded in the agents' sensitivity to changes in the demand. With regard to the resource allocation problem, this characteristic increases the system's efficiency in terms of costs and avoids the top-down propagation of deviations within the hierarchical system structure.

Publication. The concepts and results outlined in this chapter have been published in Anders et al. [12].

Due to the following reasons, dispatchable agents have to make reactive supply adjustments to satisfy the demand: (R1) The scheduling problem's complexity and the uncertainties involved call for rather coarse-grained schedules, which is why we create them for time steps that are multiples of $\Delta\tau$. Still, the demand has to be satisfied in a fine-grained time pattern of $\Delta t \leq \Delta\tau$ (see Section 1.2). Even if the demand the agents expect for $t_{\text{now}} + \Delta\tau$ turns out to be correct, they must not assume that it develops linearly from t_{now} to $t_{\text{now}} + \Delta\tau$. In the synchronous grid of Continental Europe, for instance, schedules are typically created with a resolution of $\Delta\tau = 15$ min, whereas imbalances have to be detected and compensated for within seconds (e.g., $\Delta t = 5$ s) to ensure the grid's stable operation [214]. Nowadays, the balance is maintained by a series of three successive interdependent control actions. While primary control underlies a globally standardized mechanism, the currently semi-automatic and centralized secondary control may be realized differently in each control area. This gives room for new approaches like the one presented in this chapter. (R2) Some agents might contribute according to the wrong scenario. Such a mistake is caused by a false expectation of how the demand will develop, e.g., from t_{now} until $t_{\text{now}} + \Delta\tau$. For instance, the agents might contribute according to the anticipated scenario, but a less probable scenario actually occurs. (R3) The system must be able to deal with unforeseen developments of the demand, i.e., even those that are not captured by any demand scenario in the

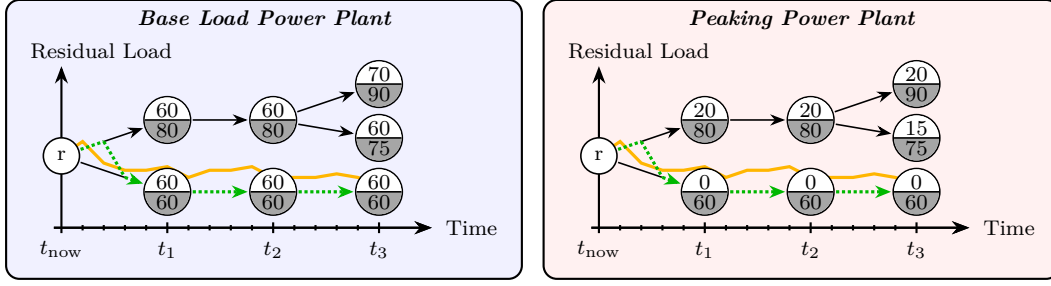


Figure 12.1: The power plants use TBST-based schedules as a blueprint for how much power to provide in which situation. For each node, the sum of the scheduled output (upper half of the nodes) of the base load and the peaking power plant matches the corresponding expected residual load (lower half of the nodes). For instance, the base load power plant should provide 60 MW and the peaking power plant 20 MW in case of a residual load of 80 MW in t_1 (here, $t_i = t_{\text{now}} + i \cdot \Delta\tau$ with $i = \{1, 2, 3\}$). While the base load power plant should provide an output that is more or less constant and independent of changes in the residual load, the peaking power plant – whose maximum output is 20 MW – should make adjustments in accordance with residual load changes. The dotted line symbolizes the development of the selection of the most suitable scenario, i.e., the scenario the power plants identified as closest to the actual residual load, which is indicated by the light curve. Between t_{now} and t_1 , the power plants reactively switch from one scenario to another after they noticed that the demand tends to be closer to 60 MW instead of 80 MW.

demand tree. Further, deviations resulting from agents that do not comply with their schedules (e.g., due to technical difficulties) have to be cushioned. Additionally, (R4) an intermediary has to reactively and locally compensate for deviations between the anticipated local demand (i.e., the “presumed *realization*” of the local demand) it reported to its superior in the course of the schedule creation and the local demand’s (actual) realization (see Section 11.2).

To hold the balance between the aggregated supply $C_V[t]$ and the demand $D[t]$, each dispatchable agent $a \in \mathcal{V}$ thus has to decide about a supply $C_a[t_{\text{next}}]$ for the next time step $t_{\text{next}} = t_{\text{now}} + \Delta t$ such that the aggregated supply $C_V[t_{\text{next}}]$ of all dispatchable agents \mathcal{V} matches $D[t_{\text{next}}]$ in t_{next} as accurately as possible (please note that we use a slightly different notation in this chapter to deal with the characteristics of reactive resource allocation). Because $D[t_{\text{next}}]$ is not known beforehand, the dispatchable agents have to determine and adjust their supply according to a *demand target* $D^T[t_{\text{next}}]$. This basically means that the agents have to compensate for the deviation between their scheduled supply for t_{next} and $D^T[t_{\text{next}}]$. With regard to specific demand scenario ω_i , we assume that the scheduled supply $S_a^{\mathcal{X}}[\omega_i, t']$ for a time step t' that was not contained in the last scheduling window \mathcal{W} is given by the linear interpolation $(S_a^{\mathcal{X}}[\omega_i, t_1] \cdot (t_2 - t') + S_a^{\mathcal{X}}[\omega_i, t_2] \cdot (t' - t_1)) / \Delta\tau$ of the scheduled contributions $S_a^{\mathcal{X}}[\omega_i, t_1]$ and $S_a^{\mathcal{X}}[\omega_i, t_2]$ for the two time steps $t_1, t_2 \in \mathcal{W}$ adjacent to t' (i.e., $t_1 < t' < t_2$). In the following, we call the problem of finding adequate reactive supply adjustments the *reactive compensation problem* (RCP).

Clearly, uncertainties require that we allow the dispatchable agents to temporarily disregard their scheduled supply. However, because schedules have been optimized with regard to the dispatchable agents’ ability to satisfy the future demand as well as with regard to economic efficiency, we want the agents to comply with their schedules as accurately as possible. So, if the current circumstances require an agent to deviate from its schedule, we have to make sure that it can quickly return to its scheduled supply in the next time steps. Note that if we did not strive for such *schedule-compliant* reactive adjustments, we could abstain from creating schedules at all. Besides schedule-compliant adjustments, the agents should respect non-functional objectives that have been regarded in the course of the schedule creation, such as satisfying the demand at low costs or avoiding the propagation of deviations to lower levels (see Sections 11.1 and 11.2), when determining reactive supply adjustments.

In Chapter 11, we explained that schedules created on the basis of Trust-Based Scenario Tress (TBST) encode strategies that indicate how many resources which dispatchable agent has to provide in which situation. To achieve the above-mentioned goals, our central idea is to make use of TBST-based schedules to proactively guide reactive supply adjustments of inert agents in order to ensure the system's efficiency, stability, and robustness. An example illustrating this principle is depicted in Figure 12.1 where the combined scheduled output of a base load and a peaking power plant has to satisfy different residual load scenarios. The figure also depicts the advantage of being able to switch to the most suitable scenario in the context of reactive supply adjustments. While the base load power plant should provide an output that is more or less constant and independent of changes in the residual load, the peaking power plant should make adjustments in accordance with residual load changes. Hence, an agent's *sensitivity* – defining when and how strongly to react to deviations between supply and demand – describes the role the agent should take on in the system. This role is predefined by the scheduling algorithm and encoded in the TBST-based schedules. An agent's role (e.g., base load or peaking power plant) depends on its control model and its production costs in relation to the other agents in the system. In general, a dispatchable agent's control model serves as an additional source of sensitivity that defines its ability to return to its scheduled supply.

The remainder of this chapter is structured as follows: In Section 12.1, we introduce a bottom-up approach to the local compensation for deviations in the hierarchical system structure. Section 12.2 outlines how TBST-based schedules proactively guide the dispatchable agents' supply adjustments, and how schedule-compliance is achieved. The empirical evaluation provided in Section 12.3 demonstrates that, when making reactive adjustments, TBST-based schedules allow the dispatchable agents to respect those objectives of the scheduling problem that can be encoded in the agents' sensitivity to changes in the demand. Related work is discussed in Chapter 13.

12.1 Reactive Compensation for Deviations in Hierarchical Systems

In the hierarchical system structure, each individual subsystem pursues the goal of balancing supply and demand by compensating for deviations locally. This avoids their propagation through the system. With respect to a subsystem \mathcal{A}_λ represented by an intermediary λ , the objective of λ 's subordinate dispatchable agents $a \in \mathcal{V}_\lambda$ is to determine their supply $C_a[t_{\text{next}}]$ for the next time step such that their aggregated supply $C_{\mathcal{V}_\lambda}[t_{\text{next}}]$ corresponds to λ 's local demand target $D_\lambda^{\mathbf{T}}[t_{\text{next}}]$. As is the case with the creation of schedules (see Section 2.2), each intermediary is thus responsible for satisfying its fraction of the overall demand. The self-organizing system structure that strives for organizations with similar ratios between uncertainty (i.e., predictability) and degrees of freedom in terms of controllability (see Section 6.3), promotes the intermediaries' ability to achieve this goal and thus prevents the propagation of uncertainties to higher levels.

Because of the dispatchable agents' limited supply and inertia, there might be, however, situations in which a subsystem is not able to provide the necessary supply on its own. In such a case, the *remaining deviation* $\Delta_\lambda^{\mathbf{R}}$ between the local demand target and the subsystem's supply has to be compensated for by "external" dispatchable agents $a \notin \mathcal{V}_\lambda$ (recall that subordinate intermediaries \mathcal{I}_λ are also contained in \mathcal{V}_λ). The remaining deviation is therefore communicated to λ 's superordinate intermediary λ' . Because λ is part of λ' 's subsystem, λ' is, in turn, responsible for compensating for $\Delta_\lambda^{\mathbf{R}}$. In case of the top-level intermediary Λ , $\Delta_\Lambda^{\mathbf{R}}$ should only be nonzero if the whole system is not able to meet the overall demand target $D^{\mathbf{T}}[t_{\text{next}}]$.

Based on the hierarchical system structure, we propose an iterative bottom-up solution to the RCP. A bottom-up procedure is especially suitable for the *local* compensation for uncertainties. An intermediary λ only communicates the remaining deviation $\Delta_\lambda^{\mathbf{R}}$ to its superordinate intermediary if it is not possible to compensate for $\Delta_\lambda^{\mathbf{R}}$ in its subtree. Although each dispatchable agent autonomously decides how much to contribute in the next time step (see Section 12.2), the process of adjusting the dispatchable agents' supply in the context of a so-called *compensation round* is coordinated by their intermediaries. Because the bottom-up propagation of deviations should be prevented, an intermediary λ that already compensated for its local deviations might decide to cover a fraction of a deviation that has

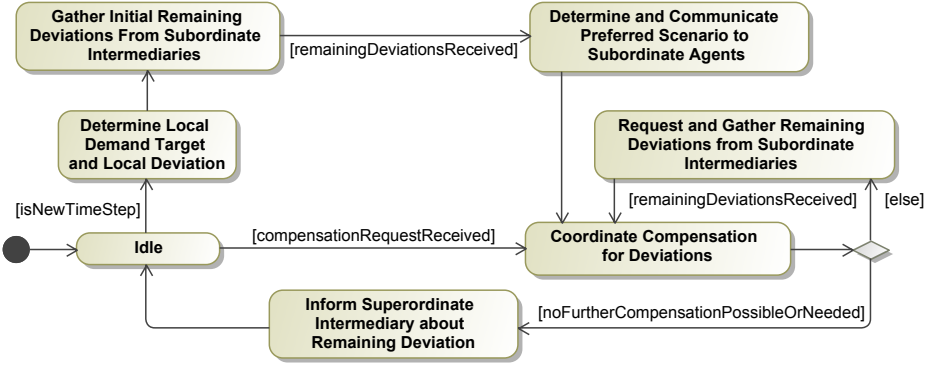


Figure 12.2: An intermediary’s basic procedure applied to compensate for deviations by means of reactive supply adjustments. The actual compensation for the deviation present in the intermediary’s subsystem is performed in the activity “Coordinate Compensation for Deviations”. Each time the intermediary calls this activity, a new *compensation round* is started.

to be cushioned by its superior λ' . Because λ , in turn, has to redistribute this fraction to its subordinates, intermediaries might perform multiple compensation rounds per time step. Figure 12.2 depicts the responsibilities of each intermediary when compensating for deviations by means of reactive supply adjustments. In the following, we explain this basic procedure step by step.

Basic Procedure

Regarding a new time step t_{now} , an intermediary λ first determines its local demand target $D_{\lambda}^{\mathbf{T}}[t_{\text{next}}]$, that is, the contribution its subordinate dispatchable agents should make in the next time step t_{next} . In case t_{now} matches the scheduling time pattern, the local demand target is determined after updating the schedules. For the sake of simplicity, we assume that an intermediary uses its current *demand to satisfy* $D_{\lambda}^{\mathbf{S}}[t_{\text{now}}]$ as target $D_{\lambda}^{\mathbf{T}}[t_{\text{next}}]$. If the period Δt between two successive time steps is short enough so that the demand does not change significantly from one time step to another, this allows for balancing supply and demand. Note that this assumption can be replaced by an arbitrary complex method that determines the local demand target.

As stated before, we want to avoid the propagation of uncertainties to higher levels. This means that each intermediary λ has to cancel out the difference between its *actual local demand* $D_{\lambda}^{\mathbf{L}}[t_{\text{now}}]$ and the *anticipated* local demand $D_{\lambda}^{\mathbf{L}}[\omega_{\alpha}, t_{\text{now}}]$ it sent to its superordinate intermediary in the course of the schedule creation (recall that we defined that the local demand originates from subordinate non-dispatchable agents as well as subordinate intermediaries in Section 2.2). The *current* demand $D_{\lambda}^{\mathbf{S}}[t_{\text{now}}]$ an intermediary has to satisfy is thus defined as the sum of its scheduled supply $S_{\lambda}^{\mathbf{S}}[\omega_{\alpha}, t_{\text{now}}]$ for the anticipated scenario ω_{α} and the above-mentioned difference between actual and anticipated local demand:

$$D_{\lambda}^{\mathbf{S}}[t_{\text{now}}] = S_{\lambda}^{\mathbf{S}}[\omega_{\alpha}, t_{\text{now}}] + (D_{\lambda}^{\mathbf{L}}[t_{\text{now}}] - D_{\lambda}^{\mathbf{L}}[\omega_{\alpha}, t_{\text{now}}])$$

For instance, if an AVPP λ communicated an anticipated local demand of $D_{\lambda}^{\mathbf{L}}[\omega_{\alpha}, t_{\text{now}}] = -5$ MW to its superior, if its scheduled supply $S_{\lambda}^{\mathbf{S}}[\omega_{\alpha}, t_{\text{now}}]$ for ω_{α} is 20 MW, and if its actual local demand $D_{\lambda}^{\mathbf{L}}[t_{\text{now}}]$ is -2 MW, the current demand this AVPP has to satisfy amounts to $D_{\lambda}^{\mathbf{S}}[t_{\text{now}}] = 20 \text{ MW} + (-2 \text{ MW} - (-5 \text{ MW})) = 23 \text{ MW}$.

Since we use $D_{\lambda}^{\mathbf{S}}[t_{\text{now}}]$ as target $D_{\lambda}^{\mathbf{T}}[t_{\text{next}}]$, subordinate dispatchable agents have to cushion the *local deviation* $\Delta_{\lambda}^{\mathbf{L}} = D_{\lambda}^{\mathbf{T}}[t_{\text{next}}] - C_{\lambda}^{\mathbf{C}}[t_{\text{next}}]$, which we define as the mismatch between $D_{\lambda}^{\mathbf{T}}[t_{\text{next}}]$ and the intermediary’s *schedule-compliant supply* $C_{\lambda}^{\mathbf{C}}[t_{\text{next}}] = \sum_{a \in \mathcal{V}_{\lambda}} C_a^{\mathbf{C}}[t_{\text{next}}]$ for the *next* time step. We define the schedule-compliant supply of a dispatchable agent $a \in \mathcal{V}$ representing a physical device as the supply that is closest to a ’s scheduled supply $S_a^{\mathbf{S}}[\omega_{\rho}, t_{\text{next}}]$ for a given scenario ω_{ρ} while not violating a ’s control model. Note that a dispatchable agent might not be able to reach its scheduled supply $S_a^{\mathbf{S}}[\omega_{\rho}, t_{\text{next}}]$ in

the next time step. That is because a dispatchable agent's current supply $C_a[t_{\text{now}}]$ does not necessarily equal its scheduled supply $S_a^{\mathcal{T}}[\omega_\rho, t_{\text{now}}]$ as a result of the reactive compensation for deviations. The concept of the schedule-compliant supply helps to reduce the frequency of such situations. Note that an intermediary λ assumes that subordinate intermediaries $\lambda' \in \mathcal{I}_\lambda$ stick to their scheduled supply for the anticipated scenario ω_α , i.e., $C_{\lambda'}^{\mathcal{C}}[t_{\text{next}}] = S_{\lambda'}^{\mathcal{T}}[\omega_\alpha, t_{\text{next}}]$ (an explanation for this assumption follows).

Although we have to allow the dispatchable agents to deviate from their schedules in order to balance supply and demand, we want them to stick to their schedules as closely as possible in order to increase the system's efficiency. Recall that TBST-based schedules prescribe how much a dispatchable agent should contribute in which situation in order to achieve the system's goals (i.e., demand satisfaction and low costs). These are encoded in the scheduling problem's objective function (see Equation (11.1a)). Because the deviation between the actual demand and a demand scenario contained in the schedule might only be of temporary nature, the system's efficiency increases if the dispatchable agents are able to quickly return to their scheduled supply. The ability to return to a scheduled supply can be expressed as the time a dispatchable agent needs to adjust its actual supply until contributing as scheduled. In case of a constant maximal rate of change ΔC_a , this time corresponds to the ratio between the schedule deviation $|S_a^{\mathcal{T}}[\omega_\rho, t_{\text{next}}] - C_\lambda^{\mathcal{C}}[t_{\text{next}}]|$ and ΔC_a . *Fairness* is regarded in the way that we want an intermediary's subordinate dispatchable agents to be equally capable of returning to their scheduled supply (i.e., the variance of the time needed to return to the scheduled supply should be small).

When compensating for deviations, intermediaries can take advantage of TBST-based schedules by switching to the scenario ω_ρ that represents the current situation as well as possible (instead of sticking to a predetermined scenario until TBSTs and schedules are recalculated). Hereinafter, we call ω_ρ the *preferred scenario*. It is defined as the scenario that – with regard to last n_{ω_ρ} time steps – comes closest to the actual local demand $D_\lambda^{\mathbf{L}}$. By laying stress on newer measurements, intermediaries adapt to new situations in short time. For each dispatchable agent, the preferred scenario specifies the scheduled supply $S_a^{\mathcal{T}}[\omega_\rho, t_{\text{next}}]$ it should be able to return to. Intermediaries identify and inform their subordinate dispatchable agents about the preferred scenario before starting the actual compensation for deviations.

As we regard a bottom-up approach, an intermediary λ does not start the compensation for its own local deviation $\Delta_\lambda^{\mathbf{L}} = D_\lambda^{\mathbf{T}}[t_{\text{next}}] - C_\lambda^{\mathcal{C}}[t_{\text{next}}]$ until all its subordinate intermediaries $\lambda' \in \mathcal{I}_\lambda$ have solved their subsystems' RCP. Once a subordinate intermediary λ' has solved the RCP, it informs λ about its remaining deviation $\Delta_{\lambda'}^{\mathbf{R}}$. As soon as all remaining deviations are available, λ initiates the compensation for $\Delta_\lambda^{\mathbf{L}}$ as well as for the remaining *subordinate deviation* $\Delta_\lambda^\downarrow = \sum_{\lambda' \in \mathcal{I}_\lambda} \Delta_{\lambda'}^{\mathbf{R}}$. λ 's subordinate dispatchable agents compensate for the deviation by iteratively adjusting their supply $C_a[t_{\text{next}}]$ for t_{next} (see Section 12.2 for more details). The compensation is completed if the deviation is dissolved (i.e., $\Delta_\lambda^{\mathbf{R}} = 0$) or no further adjustments that improve the compensation by more than a given threshold can be made. Note that if all subordinate intermediaries can deal with their subsystems' uncertainties on their own, λ is not affected by the uncertainties they have to cope with and $\Delta_\lambda^\downarrow = 0$. In such a situation, λ only has to deal with uncertainties originating from its own local demand and those stemming from its superordinate intermediary. From λ 's perspective, the subsystems of these intermediaries thus appear to be balanced. Because subordinate intermediaries inform λ about their individual remaining deviation $\Delta_{\lambda'}^{\mathbf{R}}$, λ assumes that the schedule-compliant supply $C_{\lambda'}^{\mathcal{C}}[t_{\text{next}}]$ of a subordinate intermediary λ' equals its scheduled supply $S_{\lambda'}^{\mathcal{T}}[\omega_\alpha, t_{\text{next}}]$ for the anticipated scenario ω_α . Given that $\Delta_\lambda^{\mathbf{L}} = D_\lambda^{\mathbf{T}}[t_{\text{next}}] - \sum_{a \in \mathcal{V}_\lambda} C_a^{\mathcal{C}}[t_{\text{next}}]$ and $\mathcal{I}_\lambda \subset \mathcal{V}_\lambda$, this is necessary to prevent λ from reverting compensations already made by its subordinate intermediaries.

A subordinate intermediary $\lambda' \in \mathcal{I}_\lambda$ can contribute to the compensation for λ 's deviation by committing itself to a supply adjustment. λ' regards such a supply adjustment as a deviation $\Delta_{\lambda'}^\uparrow$, it covers for λ . To ensure that ultimately a physical device stands in for this deviation, λ' has to redistribute $\Delta_{\lambda'}^\uparrow$ to its own subordinate dispatchable agents. Having completed the compensation for its deviation, λ therefore asks its subordinate intermediaries to compensate for their supply adjustments $\Delta_{\lambda'}^\uparrow \neq 0$. Since subordinate intermediaries represent independent subsystems, reactive supply adjustments can be determined in parallel, which is advantageous in terms of runtime performance. If no such supply adjustments $\Delta_{\lambda'}^\uparrow \neq 0$ have been made, λ informs its superordinate intermediary about the remaining deviation $\Delta_\lambda^{\mathbf{R}}$.

Formalization of the Reactive Compensation Problem

The RCP each intermediary λ and its subordinate dispatchable agents \mathcal{V}_λ have to solve can be formalized as follows:

$$\begin{aligned} & \underset{C_a[t_{\text{next}}]}{\text{minimize}} && \left| \Delta_\lambda - \sum_{a \in \mathcal{V}_\lambda} (C_a[t_{\text{next}}] - C_a^{\mathbf{C}}[t_{\text{next}}]) \right| \end{aligned} \quad (12.1a)$$

$$\begin{aligned} & \text{subject to} && \forall a \in \mathcal{V}_\lambda : \\ & && \exists [x, y] \in L_a^{t_{\text{next}}} : x \leq C_a[t_{\text{next}}] \leq y, \end{aligned} \quad (12.1b)$$

$$\vec{S}_a^{\min}(C_a[t_{\text{now}}]) \leq C_a[t_{\text{next}}] \leq \vec{S}_a^{\max}(C_a[t_{\text{now}}]), \quad (12.1c)$$

$$\text{with} \quad \Delta_\lambda = \Delta_\lambda^{\mathbf{L}} + \Delta_\lambda^\downarrow + \Delta_\lambda^\uparrow$$

In this optimization problem, the supply $C_a[t_{\text{next}}]$ of subordinate dispatchable agents $a \in \mathcal{V}_\lambda$ are the decision variables. Starting with the schedule-compliant supply $C_a^{\mathbf{C}}[t_{\text{next}}]$, the agents' goal is to compensate for the sum Δ_λ of all relevant deviations as accurately as possible. In this process, each participating agent stipulates its supply $C_a[t_{\text{next}}]$ for the next time step t_{next} by itself. As before, Equations (12.1b) and (12.1c) ensure that a 's supply adjustment adheres to its control model, i.e., its feasible supply ranges $L_a^{t_{\text{next}}}$ and its additional inertia functions \vec{S}_a^{\min} and \vec{S}_a^{\max} . The sum $\Delta_\lambda = \Delta_\lambda^{\mathbf{L}} + \Delta_\lambda^\downarrow + \Delta_\lambda^\uparrow$ of all relevant deviations is composed of λ 's local deviation $\Delta_\lambda^{\mathbf{L}}$, the subordinate deviation $\Delta_\lambda^\downarrow$, and the deviation Δ_λ^\uparrow that λ covers for its superior. In case of the top-level intermediary Λ , we always have $\Delta_\Lambda^\uparrow = 0$. Deviations introduced by dispatchable agents that cannot comply with their scheduled supply, e.g., due to technical difficulties, can be taken into account simply by adding them to Δ_λ . If t_{next} matches the scheduling time pattern, it is also possible to try to reactively compensate for the remaining demand that could not be redistributed in the course of an intermediary's schedule creation by adding it to Δ_λ .

It is not surprising that this optimization problem is very similar to the actual resource allocation problem that has to be solved by the system (see Equation (1.2)). By rewriting Equation (12.1a) as $|(\Delta_\lambda + \sum_{a \in \mathcal{V}_\lambda} C_a^{\mathbf{C}}[t_{\text{next}}]) - \sum_{a \in \mathcal{V}_\lambda} C_a[t_{\text{next}}]|$, the RCP's objective function is equivalent to the definition of the demand violation Δ whose minimization is the primary objective of the actual resource allocation problem (see Equation (1.2c)). Note that we do not explicitly incorporate the minimization of the costs into the RCP, though. That is because the agents base their reactive supply adjustments on their existing schedules that have been optimized with regard to costs in the course of the schedule creation.

In the following section, we explain how the dispatchable agents \mathcal{V}_λ compensate for Δ_λ by iteratively adjusting their supply $C_a[t_{\text{next}}]$ on the basis of their TBST-based schedule and the preferred scenario determined by their intermediary λ .

12.2 Determining Proactively Guided Reactive Supply Adjustments

As stated at the beginning of this chapter, we want that a dispatchable agent's supply adjustments are guided by its schedule and its controllability to increase the system's efficiency and to promote its ability to satisfy the future demand. In other words, schedules and control models should define a dispatchable agent's individual and situation-specific sensitivity to deviations. Schedules and control models thus serve as a source of *inter-agent variation* that can reduce the number of iterations needed to solve the RCP [8, 47].¹

¹In [8], we investigated the influence of *inter-agent variation* on the system's behavior when solving the resource allocation problem in a fully reactive and distributed manner, i.e., without incorporating schedules. Inter-agent variation was found in the agents' *sensitivity*, defining when and how strongly to react to deviations between demand and supply. Our investigations showed that the distribution of sensitivity is crucial to the agents' ability to solve the resource allocation problem in a specific situation (i.e., system state). Too little variation leads to oscillations, whereas too much variation to slow convergence. While we assumed that the distribution of sensitivities was predefined and a static property of the system, we found out that an adequate amount of variation significantly reduces the number of messages needed to solve the resource allocation problem.

For this reason, we assume that, in each time step that matches the schedule time pattern $\Delta\tau$, every dispatchable agent holds a schedule that is valid for a time span $H \geq 2 \cdot \Delta\tau$. Recalling that $\Delta\tau$ is the schedules' resolution and schedules have to be recalculated at least after the time span H elapsed, this constraint ensures that the dispatchable agents' reactive adjustments can be guided by their schedules. For instance, consider a time step t that is a multiple of $\Delta\tau$ and a dispatchable agent $a \in \mathcal{V}$. If, in time step $t - \Delta\tau$, a stipulates its supply $C_a[t]$ for time step t , assuming $H \geq 2 \cdot \Delta\tau$ allows a to take its scheduled supply for time $t + \Delta\tau$ into account. That is because, in time step $t - \Delta\tau$, a 's schedule was at least valid until time step $t - \Delta\tau + H \geq t + \Delta\tau$. For convenience, we assume that there have not been any demand violations in the course of schedule creations.

With regard to the procedure described in Section 12.1, dispatchable agents compensate for deviations by adjusting their supply for the next time step in an iterative process. In each iteration, the intermediary coordinating the compensation round updates and informs its subordinate dispatchable agents about the remaining deviation $\Delta_\lambda^{\mathbf{R}}$. The subordinate dispatchable agents then concurrently and independently of each other decide about a supply adjustment, i.e., they decide about a *new supply* $C_a^{\mathbf{N}}$. Because we allow $C_a^{\mathbf{N}}$ to contradict the dispatchable agent's control model (see Equations (12.1b) and (12.1c)), the closest feasible supply to $C_a^{\mathbf{N}}$ replaces $C_a[t_{\text{next}}]$ at the end of an iteration. That way, we guarantee that $C_a[t_{\text{next}}]$ complies with the agent's control model. Afterwards, the dispatchable agents inform their superordinate intermediary λ about $C_a[t_{\text{next}}]$ which, in turn, starts a new iteration in which the intermediary updates the remaining deviation according to the supply adjustments. The way a dispatchable agent adjusts its supply, and which information influences this adjustment depends on the iteration as well as whether it is the first compensation round in t_{now} or not. Recall that a new compensation round is started by the agent's intermediary λ in each new time step or if it received a compensation request from its superior (see Figure 12.2). This bottom-up procedure in which each intermediary tries to compensate for its local deviation $\Delta_\lambda^{\mathbf{L}}$, the subordinate deviation $\Delta_\lambda^\downarrow$, or the deviation Δ_λ^\uparrow that λ covers for its superior, while avoiding the propagation of deviations to higher levels, can be regarded as a "YoYo optimization style" – which is reminiscent of the principle of YoYo design [210].

In each compensation round, the iterative compensation for deviations is split into two phases: While the agent aims for schedule-compliant supply adjustments in the first phase, it increases its sensitivity in an escalating manner in the second phase. Figure 12.3 summarizes an agent's different possibilities for adjusting its supply.

Dealing with Overreactions

In general, if the dispatchable agents' total possible supply adjustment exceeds the remaining deviation $\Delta_\lambda^{\mathbf{R}}$, there is the chance of an overreaction that leads to a remaining deviation of the opposite sign in the next iteration. In this context, we define the *need* as the percentage of the last total supply adjustment $\Delta c_{\mathcal{V}_\lambda}^{\text{prev}} = \Delta_\lambda^{\mathbf{R}} - \Delta_\lambda^{\mathbf{R},\text{prev}}$ that would have been necessary to cancel out the remaining deviation $\Delta_\lambda^{\mathbf{R},\text{prev}}$ of the previous iteration (i.e., $\Delta_\lambda^{\mathbf{R},\text{prev}} = \Delta c_{\mathcal{V}_\lambda}^{\text{prev}} \cdot \text{need}$):

$$\text{need} = 1 + \frac{\Delta_\lambda^{\mathbf{R}}}{\Delta_\lambda^{\mathbf{R},\text{prev}} - \Delta_\lambda^{\mathbf{R}}}$$

Note that the denominator equals $-\Delta c_{\mathcal{V}_\lambda}^{\text{prev}}$. For instance, if we have $\Delta_\lambda^{\mathbf{R},\text{prev}} = -20$ MW as remaining deviation for the previous and $\Delta_\lambda^{\mathbf{R}} = 10$ MW for the current iteration, we obtain $\text{need} = \frac{2}{3}$.

Dispatchable agents notice the presence of an overreaction if the sign of the remaining deviation changes from one iteration to another. In such a situation, we have $0 < \text{need} < 1$. A dispatchable agent $a \in \mathcal{V}_\lambda$ reacts to such a situation by adjusting its supply by $\Delta c_a = (\text{need} - 1) \cdot \Delta c_a^{\text{prev}}$, where Δc_a^{prev} is a 's last supply adjustment. If all dispatchable agents' resulting supplies abide by their control models (i.e., if $\forall a \in \mathcal{V}_\lambda : C_a[t_{\text{next}}] = C_a^{\mathbf{N}}$), $\Delta_\lambda^{\mathbf{R}}$ will be completely absorbed. Regarding the example given above, we have $\sum_{a \in \mathcal{V}_\lambda} \Delta c_a^{\text{prev}} = \Delta_\lambda^{\mathbf{R}} - \Delta_\lambda^{\mathbf{R},\text{prev}} = 30$ MW for the agents' aggregated supply adjustment in the previous iteration so that $\Delta_\lambda^{\mathbf{R}} = 10$ MW will be absorbed if the agents make an aggregated supply adjustment of $\sum_{a \in \mathcal{V}_\lambda} \Delta c_a = (1 - \text{need}) \cdot \sum_{a \in \mathcal{V}_\lambda} \Delta c_a^{\text{prev}} = \frac{1}{3} \cdot 30 \text{ MW} = 10 \text{ MW}$. Note that the remaining deviation $\Delta_\lambda^{\mathbf{R}}$ is the only information that is shared in the subsystem of intermediary λ .

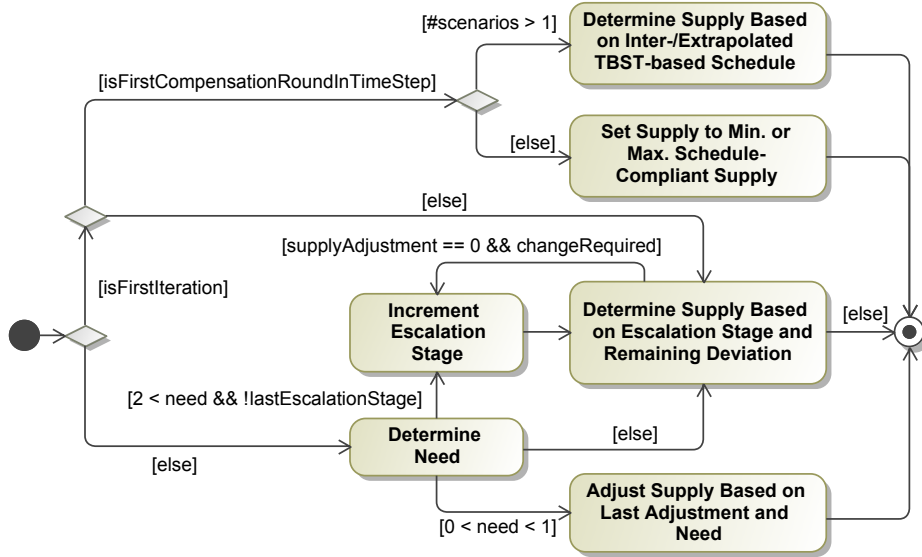


Figure 12.3: An agent's basic procedure when participating in a compensation for deviations. The decision about the future supply is based on the demand target (if available), the deviation that has to be compensated for, the scheduled supply for different demand scenarios, and the agent's control model. Except for the first iteration of the first compensation round in t_{now} , all adjustments are based on the remaining deviation. In the second phase of the iterative compensation for deviations, an agent's sensitivity escalates from one iteration to another. In the last escalation stage, adjustments are not necessarily schedule-compliant and only restricted by the agent's control model.

Scenario-Based Adjustment of the Supply

In the first compensation round in t_{now} , intermediaries compensate for the local deviation $\Delta_{\lambda}^{\text{I}}$ and the initial subordinate deviation $\Delta_{\lambda}^{\downarrow}$. In the first iteration of the first compensation round, subordinate dispatchable agents decide about their supply for t_{next} on the basis of a demand $D_{\lambda}^* = D_{\lambda}^{\text{T}}[t_{\text{next}}] + \Delta_{\lambda}^{\downarrow}$ if their schedule contains more than one scenario. In such a case, a dispatchable agent simply sets C_a^{N} to the *inter-/extrapolated scheduled supply* C_a^{I} of the two demand scenarios $D_{\lambda}[\omega_1, t_{\text{next}}]$ and $D_{\lambda}[\omega_2, t_{\text{next}}]$ that are closest to D_{λ}^* . If D_{λ}^* is between two scenarios (in this case, the supply is interpolated), we define $D_{\lambda}[\omega_1, t_{\text{next}}]$ and $D_{\lambda}[\omega_2, t_{\text{next}}]$ as the closest demand scenario above and below D_{λ}^* , respectively. When adjusting the supply, the gradient G_a , which is the ratio between the scenario-based change in supply and the absolute difference between the demand scenarios, serves as a source of sensitivity:

$$G_a = \frac{S_a^{\text{T}}[\omega_1, t_{\text{next}}] - S_a^{\text{T}}[\omega_2, t_{\text{next}}]}{|D_{\lambda}[\omega_1, t_{\text{next}}] - D_{\lambda}[\omega_2, t_{\text{next}}]|}$$

With regard to time step t_1 in Figure 12.1 and a demand $D_{\lambda}^* = 67 \text{ MW}$, the base load power plant a_1 obtains the gradient $G_{a_1} = \frac{60 \text{ MW} - 60 \text{ MW}}{80 \text{ MW} - 60 \text{ MW}} = 0$, whereas the peaking power plant a_2 obtains $G_{a_2} = \frac{20 \text{ MW} - 0 \text{ MW}}{80 \text{ MW} - 60 \text{ MW}} = 1$.

Based on its gradient G_a , a dispatchable agent determines its inter-/extrapolated scheduled supply $C_a^{\text{I}}[t_{\text{next}}]$ as follows (if D_{λ}^* is not between two scenarios, we define that $D_{\lambda}[\omega_1, t_{\text{next}}]$ is closer to D_{λ}^* than $D_{\lambda}[\omega_2, t_{\text{next}}]$ and the supply is extrapolated):

$$C_a^{\text{I}} = S_a^{\text{T}}[\omega_2, t_{\text{next}}] + G_a \cdot |D_{\lambda}^* - D_{\lambda}[\omega_2, t_{\text{next}}]|$$

Again, assuming a demand $D_{\lambda}^* = 67 \text{ MW}$ for t_1 in Figure 12.1, the base load power plant a_1 should provide an output of $C_{a_1}^{\text{I}} = 60 \text{ MW} + 0 \cdot |67 \text{ MW} - 60 \text{ MW}| = 60 \text{ MW}$, whereas the peaking power plant a_2 should deliver $C_{a_2}^{\text{I}} = 0 \text{ MW} + 1 \cdot |67 \text{ MW} - 60 \text{ MW}| = 7 \text{ MW}$.

The inter- / extrapolated supply $C_a^{\mathbf{I}}$ is used as new supply $C_a^{\mathbf{N}}$. If the inter- / extrapolated supplies of all dispatchable agents in the subsystem comply with their control models, the deviation completely dissolves after the first iteration (cf. the example above). Of course, it is possible that $C_a^{\mathbf{N}}$ does not comply with the agent's control model. In such a situation, the agent provides a contribution that comes closest to $C_a^{\mathbf{I}}$. In any case, TBST-based schedules are a good indicator of how much to contribute in a specific situation. In the above-given example, the base load power plant – whose supply should not change with the demand in different scenarios – should also provide a very similar supply in situations not captured in its TBST-based schedule (i.e., its sensitivity is low). The peaking power plant, on the other hand, has to scale its supply with the demand (i.e., its sensitivity is high). As stated at the beginning of this chapter, that is why an agent's sensitivity describes the role it should take on when solving the RCP. This role is predefined by the scheduling algorithm and encoded in TBST-based schedules. An agent's role depends on its control model and its production costs in relation to the other agents in the system. Ultimately, this allows the system to make reactive adjustments with respect to the scheduling problem's objectives. Note that schedule-based supply adjustments might lead to situations in which some dispatchable agents increase and others decrease their supply. This might, however, be necessary to be able to satisfy the future demand. To ensure progress in the compensation for deviations, we make certain that all dispatchable agents either increase or decrease their supply in all subsequent iterations and compensation rounds in t_{now} .

Schedule-Compliant Supply Adjustments

If the demand is above or below all scenarios, an agent's inter- / extrapolated supply $C_a^{\mathbf{I}}$ is limited to the minimum or maximum schedule-compliant value (see Section 12.1). Here, the schedule-compliant supply is determined with respect to the agent's preferred supply $S_a^{\mathbf{I}}[\omega_\rho, t_{\text{next}} + \Delta t]$ (that is, the contribution it should make according to the preferred scenario ω_ρ) in the time step $t_{\text{next}} + \Delta t$ after the next time step t_{next} . This procedure ensures that dispatchable agents are only at risk of not being able to deliver their scheduled supply in $t_{\text{next}} + \Delta t$ if necessary. Further, if only a single scenario is available in the first iteration of the first compensation round in t_{now} , dispatchable agents use their minimum or maximum schedule-compliant supply as $C_a^{\mathbf{N}}$, whichever is more appropriate in the given situation. While this measure is likely to cause an overreaction that is damped in the subsequent iteration (cf. the need-based procedure introduced at the beginning of this section), it is fair since the dispatchable agents' are equally able to return to their preferred supply in $t_{\text{next}} + \Delta t$. In these situations, a dispatchable agent's sensitivity (i.e., its role in the course of the compensation) is defined by its control model and further subject to its preferred supply in $t_{\text{next}} + \Delta t$.

Escalation of Sensitivity

In all following iterations or compensation rounds in t_{now} , dispatchable agents increase their sensitivity in an escalating manner. That way, dispatchable agents ensure that they make progress in solving the RCP while adhering to their schedules as accurately as possible.

The escalation stages range from schedule-based reactions, over the full exploitation of the schedule-compliant supply, to exclusively control-model-based decisions that do not take the schedule into account. We only present the basic idea of some of these stages in this chapter. Each stage specifies a factor that is used to scale the supply adjustment proportionally to the given deviation. In the first escalation stage, for example, we use the average gradient (cf. "Scenario-Based Adjustment of the Supply") of adjacent scenarios in the schedule. An example of an exclusively control-model-based decision is to make use of an agent's maximal rate of change. If $need > 2$ (cf. "Dealing with Overreactions") less than half of the last iteration's deviation was compensated. In such a situation, a dispatchable agent switches to the next escalation stage to increase its sensitivity. In case the deviation has not changed from one iteration to another, each dispatchable agent switches to the next higher escalation stage that leads to a supply change (cf. "supplyAdjustment == 0" in Figure 12.3). Note that this stage might differ from one agent to another. To promote schedule compliance, except for the last escalation stage, it is ensured that the supply is schedule-compliant with respect to $t_{\text{next}} + \Delta t$ (cf. "Schedule-Compliant Supply

Adjustments”). Finally, in the last escalation stage, dispatchable agents adjust their supply as much as possible, i.e., they set their supply to the feasible value closest to $\vec{S}_a^{\min}(C_a[t_{\text{now}}])$ or $\vec{S}_a^{\max}(C_a[t_{\text{now}}])$. While schedule compliance is not ensured in the last stage, such a reaction is very likely to cause an overreaction that is compensated for in the subsequent iteration on the basis of the *need*, which, at least, yields fair adjustments with regard to the exhausted dispatchable agents’ flexibility. However, before such drastic measures are taken, dispatchable agents stay in a waiting stage as long as other agents can compensate for deviations without infringing schedule compliance. If no further schedule-compliant supply adjustments can be made, the agents switch to the last escalation stage (cf. “changeRequired” in Figure 12.3).

12.3 Evaluation in the Decentralized Power Management Case Study

In the context of our case study, we evaluate the TBST-based reactive compensation for deviations between supply and demand in five different settings. These mainly differ in the information dispatchable power plants use to determine their sensitivities and supply adjustments. Where not stated otherwise, presented results are average values.

Test Bed

We base our evaluation on a hierarchical structure of AVPPs, consisting of 173 dispatchable and 350 non-dispatchable power plants of different types (hydro, biofuel, gas power plants as well as solar plants and wind generators). We use a hierarchy that was established by the power plants in a self-organized manner in preliminary tests. It contains 20 AVPPs and is of height 5.

As before, each power plant is modeled as an individual agent and non-dispatchable consumers are represented by a single agent that is assigned to the top-level AVPP. The demand, i.e., the residual load, originates from weather-dependent power plants and the consumers. We use real world data for the capabilities of physical power plants² (such as minimal and maximal production boundaries), the load curves³, and the simulated weather conditions⁴ influencing the output of weather-dependent power plants. Each dispatchable power plant’s inertia is defined within typical boundaries. The production costs, i.e., the unit price of electricity, range from 6.50 $\frac{\text{euro cent}}{\text{kWh}}$ to 17.50 $\frac{\text{euro cent}}{\text{kWh}}$.

The simulation environment is implemented in a sequential, round-based execution model. Each round corresponds to a specific time step $t \in \mathcal{T}$. The dispatchable power plants have to satisfy a prescribed residual load (mean value of 1.43 GW) over a period of 12 hours, corresponding to $|\mathcal{T}| = 720$ discrete time steps, each representing 1 min. Every 15 min, AVPPs update their TBSTs and create schedules for a time span of $H = 1$ hour with a resolution of $\Delta\tau = 15$ min on the basis of residual load predictions provided by non-dispatchable power plants and consumers. Each schedule thus comprises $N = 4$ time steps.

AVPPs employ the regio-central approach *RegioC* for the creation of TBST-based schedules. The corresponding optimization problems are formulated as mixed integer linear programs and solved by IBM ILOG CPLEX⁵. When creating schedules, the utmost goal is to satisfy the residual load, followed by the goal to provide electricity at a low price. We opt for RegioC in this evaluation since its schedules have a greater need for reactive supply adjustments than those of TruCAOS (see Section 11.5).

As in Chapters 5 and 11, uncertainty in residual load predictions is generated by means of Markov models that randomly modify the actual behavior of non-dispatchable power plants and consumers. To reflect sequential dependencies in the predictions’ accuracy (as is the case with the quality of weather predictions, for instance), the prediction error of a specific time step depends on prediction errors of previous time steps. The stochastic process is further designed in a way that short-term predictions

²Energymap (Bavaria), 2012: <http://www.energymap.info>, retrieved in 2012.

³LEW, 2012: <http://www.lew-verteilnetz.de>, retrieved in 2012.

⁴LfL (Bavaria), 2010: <http://www.lfl.bayern.de/agm/>, retrieved in 2012.

⁵IBM ILOG CPLEX Optimizer, Version 12.4, 2011: <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, retrieved on March 1, 2016.

are more precise than long-term predictions, and that predictions become more accurate as a future point in time approaches. Calculated over all regarded time steps, the mean prediction error was 3.57 % of the actual local residual load with a standard deviation of 5.25 %. The accuracy of predictions of non-dispatchable power plants within the same AVPP is coupled as is the case with weather predictions for different locations. The top-level AVPP is able to measure the quality of residual load predictions with an accuracy of 3.0 % of the mean local residual load recorded in the experiences that are used for the generation of the local deviation tree. All other AVPPs can perceive erroneous predictions with an accuracy of 20.0 %. Again, we use a higher resolution for the top-level AVPP because its local residual load includes the consumption and is thus much higher than the local residual load of the other AVPPs. In every time step $t \in \mathcal{T}$, power plants compensated for deviations between power production and consumption by solving the RCP as described in Sections 12.1 and 12.2.

In the following, we analyze the influence of the information the dispatchable power plants use to determine their reactive supply adjustments on runtime performance and on the quality of the compensation. To this end, we make use of five different evaluation scenarios, called *OnlyLastEscStage*, *AllEscStages*, *ScenarioBased*, *BadPrefScenario*, and *1-TBS*. In *OnlyLastEscStage*, power plants compensate for deviations exclusively by applying the *last* escalation stage (see “Escalation of Sensitivity” in Section 12.2). In *AllEscStages*, power plants make use of all escalation stages, including those that incorporate schedule-based decisions, such as the average gradient of adjacent scenarios in the schedule, and that allow for schedule-compliant adjustments (see “Schedule-Compliant Supply Adjustments” in Section 12.2). In *ScenarioBased*, power plants additionally inter-/extrapolate their output on the basis of the different demand scenarios contained in their TBST-based schedules (see “Scenario-Based Adjustment of the Supply” in Section 12.2). In *BadPrefScenario*, we force the AVPPs to choose the most inappropriate scenario as preferred scenario to investigate the mechanism’s resilience with respect to badly chosen preferred scenarios. In *1-TBS*, all escalation stages can be applied but schedules are only created for the anticipated demand scenario. For each evaluation scenario, we performed 200 simulation runs, which amounts to $200 \cdot |\mathcal{T}| = 200 \cdot 720 = 144000$ observations per evaluation scenario used to calculate mean values and standard deviations. The differences we point out in the following discussion are statistically significant using a two-sample t-test at $\alpha = 0.01$.

Results

Table 12.1 lists the results for the four evaluation scenarios in which TBST-based schedules are created. The results for the evaluation scenario *1-TBS* are not included in the table as they are very similar to those of *OnlyLastEscStage*. We highlight the most important differences at the end of this section.

Independently of a specific evaluation scenario, we observe that the compensation time per AVPP of not more than 1.40 ms as well as the maximum sequential compensation time⁶ of not more than 8.02 ms is only a fraction of the corresponding scheduling time per AVPP of 351.05 ms (standard deviation $\sigma = 345.06$ ms) and the max. seq. scheduling time of 2394.54 ms ($\sigma = 929.04$ ms). On average, AVPPs created schedules for 3.70 scenarios ($\sigma = 2.17$). The given numbers for the scheduling time per AVPP, the max. seq. scheduling time, and the number of scenarios refer to the evaluation scenario *ScenarioBased* but are more or less identical for all evaluation scenarios in which TBST-based schedules are created. This illustrates that the reactive compensation for deviations can be performed in much less time than the creation of TBST-based schedules, which is why a rather coarse-grained schedule resolution is preferred. Furthermore, it should be noted that the power plants managed to reactively dissolve all deviations in all evaluation scenarios.

In *OnlyLastEscStage*, power plants compensate for deviations exclusively by means of the last escalation stage. Hence, their reactions are not influenced by their schedules but only subject to their control models. Since this is likely to cause an overreaction that is damped in a subsequent iteration, the power plants need only 2.02 iterations on average to dissolve a deviation. Despite these overreactions, our procedure ensures that fairness – with respect to the power plants’ exhausted flexibility – is respected.

⁶We define the maximum sequential compensation time analogously to the maximum sequential scheduling time (see Equation (6.1)). The maximum sequential compensation time is thus the longest of all serial paths that result from adding the compensation times for each branch in the hierarchy.

	<i>ScenarioBased</i>	<i>AllEscStages</i>	<i>OnlyLastEscStage</i>	<i>BadPrefScenario</i>
Max. Seq. Compensation Time per Time Step [ms]	7.47 (9.88)	8.02 (10.16)	6.33 (9.66)	7.62 (10.24)
Aggr. Compensation Time per AVPP and Time Step [ms]	1.30 (2.40)	1.40 (2.49)	1.16 (2.23)	1.34 (2.49)
#Compensation Rounds per AVPP and Time Step	2.09 (1.00)	2.08 (1.01)	3.13 (1.03)	2.14 (1.01)
#Iterations per Compensation Round	3.49 (2.23)	3.86 (2.49)	2.02 (0.20)	3.56 (2.25)
#Switches of the Preferred Scenario Between Two Schedule Calculations	1.56 (0.68)	1.56 (0.69)	1.57 (0.70)	1.95 (0.71)
Local Deviation per AVPP and Time Step [%]	76.76 (27.82)	77.62 (27.46)	66.33 (27.00)	77.51 (26.93)
Subordinate Deviation per AVPP and Time Step [%]	0.48 (5.06)	0.47 (5.07)	0.50 (5.24)	0.44 (4.69)
Superordinate Deviation per AVPP and Time Step [%]	22.76 (27.49)	21.91 (27.10)	33.17 (26.81)	22.06 (26.65)
Schedule Compliant Output per Disp. Power Plant and Time Step [%]	88.32 (17.90)	88.74 (17.50)	92.04 (11.53)	86.03 (20.11)
Used Percentage of Max. Output Change per Disp. Power Plant and Time Step [%]	7.30 (8.21)	6.75 (7.71)	4.49 (2.81)	8.77 (9.71)

Table 12.1: Evaluation results for the four evaluation scenarios *ScenarioBased*, *AllEscStages*, *OnlyLastEscStage*, and *BadPrefScenario*. All results are averaged over the 200 simulation runs for each evaluation scenario. Values in parentheses denote standard deviations. The results for the evaluation scenario *1-TBS* are similar to those of *OnlyLastEscStage*. Differences are highlighted in the discussion.

This is reflected in the comparably low standard deviation of 2.81 % of the percentage of $\vec{S}_a^{\min}(C_a[t_{\text{now}}])$ or $\vec{S}_a^{\max}(C_a[t_{\text{now}}])$ power plants make use of when compensating for deviations (mean value of 4.49 %). Additionally, the relatively low standard deviation of 11.53 % of schedule-compliant outputs shows that the procedure is also fair in terms of the power plants' ability to return to their preferred scheduled output (mean value of 92.04 %). These values significantly differ from the results for the other evaluation scenarios. However, as AVPPs are usually able to change their output to a greater amount than physical power plants, this single source of sensitivity leads to a propagation of deviations to subordinate AVPPs. Regarding a single compensation round, only 0.50 % of the deviations are propagated to the superordinate AVPP on average, but 33.17 % of a deviation stems from the superordinate AVPP. This yields to a relatively high average number of 3.13 compensation rounds per AVPP and time step. To prevent this top-down propagation of uncertainties, we, in fact, included a term in the *scheduling problem's* objective function such that solutions in which AVPPs provide similar outputs for different residual load scenarios are preferred, that is, AVPPs should act as a specific type of base load power plant (see Section 11.2). Our evaluation shows that this objective is ignored when power plants decide about reactive adjustments without incorporating their schedules.

For this reason, we enable the power plants to use all escalation stages, including those that incorporate schedule-based decisions, in the evaluation scenario *AllEscStages*. Because the power plants' sensitivity is successively increased in this setting, they now need 3.86 iterations per compensation round on average. On the other hand, we notice that only 21.91 % of a deviation stems from the superordinate AVPP, which is a decrease of about 34 % compared to *OnlyLastEscStage*. This behavior confirms that reactive supply adjustments that are guided by TBSTs and schedule-based decisions allow the agents to respect those objectives of the scheduling problem that can be deduced from the created schedules. These objectives comprise those that can be encoded in a power plant's sensitivity to changes in the demand, such as achieving low prices or preventing the propagation of local uncertainties to lower levels.⁷ Moreover, the mean number of compensation rounds per AVPP and time step drops to 2.08. Although the number of iterations is larger than in *OnlyLastEscStage*, the number of sent messages is significantly decreased in

⁷Objectives that cannot be deduced from created schedules include, for example, "produce more than 60 % green energy" or "do not turn on / off the power plant more than twice a day".

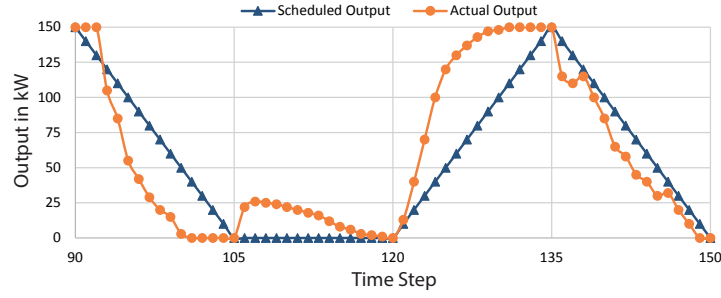


Figure 12.4: The scheduled output for the anticipated scenario as well as the actual output of a hydro power plant over a time frame of 60 min. Recorded in our simulation environment for autonomous power systems. Despite necessary reactive supply adjustments, the actual output resembles the scheduled output.

AllEscStages. That is because, in *OnlyLastEscStage*, the number of compensation rounds performed by AVPPs increases from top to bottom in the hierarchy since *OnlyLastEscStage* tends to propagate deviations to lower levels. Note that also the number of power plants increases from top to bottom. Since we ran our experiments on a single machine, our time measurements do not include communication efforts, though. As the average compensation time per AVPP is in the range of milliseconds, this decrease in the number of sent messages would improve the performance of a real distributed system (assuming that the time needed to deliver a message also ranges in the order of milliseconds).

When allowing power plants to use their TBST-based schedules to derive an inter-/extrapolated output in evaluation scenario *ScenarioBased*, we observe that, compared to *AllEscStages*, the average number of iterations is reduced (3.49 instead of 3.86), which results in a shorter runtime per AVPP of 1.30 ms and a max. seq. runtime of 7.47 ms. Although the schedule-based decisions of *ScenarioBased* and *AllEscStages* do not yield the same degree of fairness as *OnlyLastEscStage* (recall that fairness mainly results from the way power plants deal with overreactions), the principle of proactively guided reactive adjustments allows them to respect the scheduling problem’s objectives.

In *BadPrefScenario*, we forced the AVPPs to choose the most inappropriate scenario as preferred scenario. Although the power plants need slightly more iterations as well as the highest percentage of their maximal possible output change among all evaluation scenarios, they are still able to compensate for deviations. This points out the mechanism’s resilience with respect to the selection of the preferred scenario (1.95 switches between two schedule calculations). The fact that AVPPs also switched the preferred scenario at least 1.56 times on average in the other TBST-based evaluation scenarios indicates the importance of TBST-based schedules and of being able to choose the most suitable scenario at runtime.

The results for the evaluation scenario *1-TBS* in which schedules are only created for the anticipated demand scenario are very similar to those obtained in *OnlyLastEscStage*. That is because the power plants cannot make use of the information about how to behave in different demand scenarios. As is the case with *OnlyLastEscStage*, reactive adjustments are therefore only guided by the power plants’ control model. This results in a propagation of deviations to subordinate AVPPs. However, because the power plants do not only make use of the last escalation stage, they need an average of 2.67 ($\sigma = 0.36$) iterations per compensation round. The disadvantage of not taking multiple demand scenarios into account is revealed when taking a look at the resulting costs per schedule. While schedule-based reactive adjustments in *ScenarioBased* lead to mean total costs of the schedules of about 173.56 k € ($\sigma = 35.67$ k €) and *OnlyLastEscStage* causes expenses of 173.95 k € ($\sigma = 35.54$ k €), considering only a single scenario in *1-TBS* generates costs of 175.70 k € ($\sigma = 34.57$ k €). With respect to the total considered time span of 12 hours, the difference in costs between *OnlyLastEscStage* and *1-TBS* amounts to 102.72 k €. This demonstrates the financial advantage of regarding multiple scenarios when creating schedules and making reactive supply adjustments in uncertain environments. Note that the differences in costs would probably be even larger if we considered only for the first two instead of all four time steps in a schedule.

Figure 12.4 shows an example of how a hydro power plant's actual and scheduled output for the anticipated scenario developed over a time frame of 60 min. The scheduled output was stipulated for time steps in $\{90, 105, 120, 135, 150\}$. Although uncertainties necessitate deviations between the actual and the scheduled output, the characteristic of the reactively determined actual output resembles the development of the scheduled output. This illustrates our idea of schedule compliance.

Chapter Summary and Outlook

In this chapter, we showed that the combination of proactive schedule creation and reactive supply adjustments allows the dispatchable agents to effectively and efficiently solve the resource allocation problem in uncertain environments. In particular, the system has to rely on reactive supply adjustments to balance supply and demand in a fine-grained time pattern although the scheduling problem's complexity calls for rather coarse-grained schedules. Furthermore, reactive supply adjustments enable the system to deal with unforeseen developments of the demand and deviations resulting from agents that do not comply with their schedules. In this context, schedules created on the basis of Trust-Based Scenario Trees (TBSTs) proactively guide the agents' reactive adjustments since they encode how many resources the agents have to provide in which situation. Moreover, they allow the agents to switch to the most suitable schedule at runtime. Together with the agents' control model, TBST-based schedules serve as a dynamic and situation-specific source of sensitivity to deviations between supply and demand. This sensitivity defines an agent's role in the compensation for deviations. Our evaluation demonstrates that reactive adjustments guided by TBST-based schedules respect those objectives of the scheduling problem that can be deduced from created schedules. This characteristic increases the system's efficiency in terms of costs and avoids the propagation of deviations within the hierarchical system structure.

In future work, we want to investigate the on-demand recalculation of schedules, i.e., we want to delay the recalculation of schedules until it is observed that the current schedules might not adequately guide the agents' reactive supply adjustments in future. Ultimately, a shift from rigid to flexible scheduling intervals should reduce the overhead of unnecessary schedule recalculations while preserving the system's efficiency and robustness.

Related Work

Summary. This chapter provides a cumulative discussion of the related work for Chapters 10 to 12 in the context of finding robust solutions to the resource allocation problem. While we examine different related approaches to schedule creation and reactive resource allocation in general, we are especially interested in the others' approaches' ability to deal with uncertain demand and supply, and to cope with systems of large size.

Publication. The most part of the following discussion has been published in Anders et al. [8, 10, 11, 12, 14] and Kosak et al. [123].

In the domain of power systems, multiple approaches aim at solving problems similar to the scheduling problem considered in this thesis (see Section 1.2). Due to the complexity of these problems, centralized approaches – based on, for example, particle swarm optimization [84] or standard mathematical programming software [234] – can only handle small numbers of dispatchable agents, though. Our empirical evaluations in Sections 9.2, 10.5, 11.2, and 11.4 confirm that the self-organized formation of hierarchical system structures allows large systems to deal with the complexity of the scheduling problem. Based on such a hierarchical structure, the regio-central approach *RegioC* presented in Section 2.2 – in which each intermediary creates schedules by means of an instance of the centralized solver IBM ILOG CPLEX – is able to solve the scheduling problem for a large number of agents in short time. However, our evaluations also revealed that a heuristic, such as TruCAOS (see Chapter 10), significantly improves the system's runtime performance, the quality of schedules, and – because TruCAOS is able to deal with large subsystems – the stability of the system structure. Importantly, these characteristics allow TruCAOS to increase the system's robustness by creating robust solutions to the scheduling problem, which is achieved by regarding multiple possible developments of the demand, optimizing the provision of reserves, and shifting the dispatchable agents' scheduled supply into regions of higher predictability. With regard to uncertainties, Zafra-Cabeza et al. [234] look into the application of risk management to scheduling combined heat and power plants. Given a mapping from risks to countermeasures that is defined at design time, risks (e.g., the risk to violate a schedule) can be mitigated by scheduling corresponding actions, e.g., hiring more personnel or carrying out maintenance procedures. These investigations are orthogonal to those made in this thesis: Since TruCAOS's trust-based measures and its payment function incentivize compliance with schedules, the operators of power plants are encouraged to create proposals by means of the techniques presented in [234].

Apart from centralized algorithms, there are several market-based approaches related to TruCAOS, such as [121, 131, 227]). DEZENT [227] is a bottom-up market-based mechanism that operates on a predefined hierarchical structure based on the different voltage levels of the power grid. Each internal node acts in the role of a balancing group manager for a specific voltage level. Leaves correspond to physical devices, such as generators. DEZENT balances power production and consumption by enabling the agents to conclude contracts within fixed price frames that are tightened by the balancing group managers from one iteration to another. Agents that could not sell or buy electricity as desired are

forwarded to the balancing group manager on the next higher hierarchy level. DEZENT assumes that the root – which stands for the highest voltage level – is able to compensate for any remaining mismatch between supply and demand. A further approach to supply and demand matching that is based on a hierarchical system structure is PowerMatcher [121]. In PowerMatcher, physical devices constitute the leaves of a hierarchy, too. However, internal nodes only serve to aggregate subordinate supply and demand. The hierarchy’s root balances supply and demand by determining an equilibrium price, i.e., a price that establishes a market equilibrium. The equilibrium price is based on aggregated demand, aggregated supply, and price predictions. The auctioneer – in the form of the root – is thus a central component of the system. As opposed to the resource allocation problem considered in this thesis, satisfying the demand is not paramount. Unlike TruCAOS, DEZENT and PowerMatcher only create schedules for the next time step instead of multiple future ones. While this is reminiscent of the reactive compensation problem (see Chapter 12), DEZENT and PowerMatcher are not suitable for enabling dispatchable agents to make fast reactive supply adjustments that balance supply and demand. Li et al. [131] present a coordination mechanism that uses a blackboard, called stigspace, as the medium of communication between distributed energy resources in order to create schedules for multiple time steps in an iterative process. Initially, the stigspace is used to announce the demand that has to be fulfilled by the distributed energy resources. These in turn revise their schedules in order to minimize the remaining demand. The new schedule is then posted to the stigspace where the remaining demand is updated accordingly. This process is repeated until the demand is sufficiently satisfied. In this approach, agents adjust their schedules independently of each other, and every schedule posted to the stigspace is “accepted”. Obviously, this might have detrimental effects on the approach’s convergence. Above that, the method proposed by Li et al. [131] might only achieve the resource allocation problem’s primary objective, that is, to allocate resources according to a given demand. Non-functional requirements, such as the minimization of costs, are not taken into account. Our evaluations in Sections 10.5 and 11.4 revealed the importance of minimizing the costs in the course of schedule creation. The stigspace approach is not only related to TruCAOS, but also to our bottom-up procedure for reactive supply adjustments: an intermediary can be thought of as the stigspace and, within an iteration, dispatchable agents determine their supply adjustments independently of each other. However, as the agents base their decisions on schedules that have been created for multiple demand scenarios, their supply adjustments are proactively guided and thus indirectly coupled, which allows them to respect objectives of higher system levels. In contrast to our approach, DEZENT, PowerMatcher, as well as the stigspace approach do not consider the problem of uncertain supply and demand. They are thus not able to create robust schedules to the scheduling problem.

Hinrichs et al. [89] present a decentralized heuristic, called COHDA₂, for solving a combinatorial optimization problem. The algorithm is based on a neighborhood structure in which agents iteratively adjust their configuration, i.e., state, in order to achieve a global as well as individual objectives. The former can be thought of as the objectives of the resource allocation problem regarded in this thesis. In COHDA₂, every time an agent changes its configuration, it informs its neighbors about its new configuration as well as the currently known configuration of its neighbors and the neighbors’ neighbors etc. it received in a prior time step. Although each agent only has current information about its direct neighborhood, it synthesizes a complete representation of the configuration of all agents in the system over time. To decide on suitable own configurations, the agents take this locally perceived global configuration into account. As it is very likely that this configuration differs from the actual configuration of the other agents, the agents keep track of and inform their neighbors about the best configuration they have achieved so far. While COHDA₂ solves a combinatorial optimization problem in a truly decentralized manner, the quality of its solutions highly depends on the topology of the overlay network defining the agents’ neighborhood. However, COHDA₂ does not consider uncertainties and is thus not able to aim for robust solutions to the scheduling problem. In [156], COHDA₂ is extended to recalculate schedules for the members of a virtual power plant as soon as an incident is detected that leads to a deviation between scheduled and actual supply. Instead of recalculating schedules, our approach to reactive supply adjustments exploits the information provided by robust solutions to the scheduling problem, which allows for balancing supply and demand in very short time.

As outlined in Chapters 5 and 11, stochastic programming [198] and online stochastic optimization [83]

are well-known frameworks for solving optimization problems in uncertain environments based on a set of scenarios. Pappala and Erlich [162] introduce a particle swarm optimizer, i.e., a centralized population-based metaheuristic, that solves multi-stage problems in the context of schedule creation. While their algorithm thus takes account of the scenario tree's structure (see Figure 11.2a), it has only been evaluated in a system consisting of 11 power plants; no statements regarding runtime performance were made. Due to the computational complexity of solving multi-stage stochastic programs with a centralized optimizer [197], most approaches making use of these frameworks use a two-stage stochastic program as approximation (cf. [41, 71, 178]). Complexity is reduced because two-stage problems treat a given scenario tree as a fan of unrelated scenarios (see Figure 11.2b). A variant of two-stage problems presented in [193] abolishes recourse actions for the next relevant time step by representing the uncertainties by a single expected value (see Figure 11.2c). The expected value is the average of the root's children in the given scenario tree. Evidently, regarding an expected value does not allow for encoding strategies for how to react to different situations. Beyond that, the method presented by Scott et al. [193] does not take the scenarios' probabilities into account. By contrast, our approach to robust schedule creation exploits the scenario trees' structure as well as the scenarios' probability of occurrence. Considering multiple scenarios – also in the first stage – allows for switching to the most suitable scenario at runtime, which proved to be beneficial to the system's robustness and efficiency (cf. the evaluation results presented in Sections 5.3, 11.2, and 12.3). Furthermore, we demonstrated that scenario-tree-based schedules are an enabler for efficient and effective self-adaptation when reactively balancing supply and demand (see Section 12.3). As opposed to these solutions, our approach to deal with the complexity of multi-stage problems is based on self-organized hierarchical problem decomposition in combination with an auctioned-based mechanism that enables all dispatchable agents to actively participate in the process of schedule creation.

While Ruiz et al. [178] also optimize the provision of reserves to increase the system's robustness, reserves are implemented as hard constraints instead of an additional objective of the scheduling problem. As our evaluation in Section 11.5 shows, hard constraints are usually too restrictive. Although the centralized approach presented by Ruiz et al. [178] cannot be applied to hierarchical systems, their results confirm our observation that the combination of scenario-based scheduling and the optimized provision of reserves leads to robust solutions.

Ruthe et al. [179] regard the problem of matching uncertain demand and supply in power markets. The goal is to determine a stochastic price vector, i.e., a market clearing price, that balances the expected demand and the expected supply. This is in contrast to the scheduling problem considered in this thesis where schedules have to be stipulated in a way that the aggregated supply meets the given demand while minimizing the resulting costs.

Ströhle et al. [208] present a centralized scheduling heuristic for solving a combinatorial resource allocation problem under uncertain power consumption and renewable supply. The task is to schedule non-preemptive jobs in the form of deferrable loads. These jobs are characterized by a value, a consumption rate, a duration, and a deadline, among others. Future jobs are uncertain so that the centralized scheduler only has a probability distribution of future jobs and their properties. In [208], the goal is to maximize the social welfare, i.e., the net profit given as the difference between the total value of scheduled jobs and the costs of supply. A further important difference to the scheduling problem regarded in this thesis is that, in each time step, an unlimited amount of conventionally generated supply is available. This supply can be provided at fixed costs in order to balance power production and consumption. Similar to TruCAOS's price-performance filtering, a greedy algorithm creates schedules according to the jobs' price-performance ratio. Uncertainty regarding future renewable supply is modeled by means of scenarios which are sampled from a fixed probability distribution instead of being learned at runtime. Although each scenario has a certain probability of occurrence that is used to maximize the expected social welfare, only a set of *unrelated* scenarios is considered. As opposed to [208], our approach to robust schedule creation incorporates the tree's structure into the optimization problem. By scheduling the supply for nodes (which might be part of multiple scenarios), we ensure that the agents cannot only provide the scheduled output in $t_{\text{now}} + \Delta\tau$ but also in subsequent time steps (note that there might be a branch following a node in $t_{\text{now}} + \Delta\tau$). This characteristic allows the agents to delay their decision which demand scenario to choose, i.e., their assumption in which direction the

demand develops, until the time step a branch in the tree has to be taken (see Chapter 11). In addition, TruCAOS accepts proposals in a way that increases the predictability of dispatchable agents.

In addition to *unintentional* uncertainties originating from weather-dependent power plants, Ströhle et al. [208] employ the principle of *mechanism design* [59] to address the issue of *intentional* uncertainties. The field of mechanism design studies how a system has to work in order to *incentivize* its self-interested, strategic, and individually rational participants to tell the truth, i.e., to behave benevolently. Intentional misbehavior can be ascribed to agents that lie about some private information needed to decide about an adequate allocation, such as the cost or probability of performing a task successfully [59, 165]. As opposed to unavoidable misbehavior, uncertainties originating from intentional misbehavior can be avoided. Employing the techniques of mechanism design can guarantee efficiency (maximization of the agents' overall utility), individual rationality (the agents' utility of participating in the scheme is non-negative), and incentive compatibility (the agents are best off revealing their true type) [60]. The latter property is of particular interest in open systems when agents have to be incentivized to disclose their private information needed to make decisions. As the approach presented by Ströhle et al. [208] is incentive compatible, dispatchable agents are incentivized to report their true type, i.e., their job's true valuation, duration, etc.

In the context of supply and demand matching in smart grids, the ideas of mechanism design have been adopted to various market-based approaches in which pricing mechanisms prevent agents from gaming the system (e.g., [48, 220]). In such settings, one has to be careful on which assumptions the presented approaches proceed to achieve incentive compatibility. Chalkiadakis et al. [48], for example, employ a pricing mechanism that incentivizes power plants to provide accurate estimates about their production. However, this mechanism is based on the assumption that there is a global unit price of electricity that can be used to determine the agents' payment. We must not make this assumption for the problem considered in this thesis where each dispatchable agent has an individual cost function. For this reason, TruCAOS makes use of the mean average costs of accepted proposals to lower the reward of a dispatchable agent that did not comply with its schedule. Dash et al. [60] achieves incentive compatibility by means of a payment function that penalizes supply that is *lower* than scheduled or predicted (in case of weather-dependent power plants). This measure prevents dispatchable agents from intentionally overestimating their supply, and additionally reduces avoidable uncertainties, e.g., the risk that (non-)dispatchable agents overestimate their supply if it could unintentionally turn out to be lower than expected. Underestimations that also lead to imbalances between supply and demand are not penalized. In [60], the goal is thus merely to *cover* the demand instead of balancing supply and demand. In contrast to [60], TruCAOS is able to create schedules for multiple time steps and its payment function takes both positive as well as negative deviations from promised supply into account. Because we cannot make such assumptions, TruCAOS is not incentive compatible. However, as Dash et al. [60] shows, it is still possible to use penalty schemes to increase the agents' risk that providing false reports or promises that cannot be kept is detrimental to their utilities. TruCAOS's payment function follows this idea so that dispatchable agents are incentivized to comply with their promised supply (see Section 10.4). Moreover, since a dispatchable agent's predictability influences the price-performance ratio of its proposals, trustworthy agents can charge higher prices for a contribution than untrustworthy agents.

Although the approaches presented in [48, 60, 220] focus on preventing intentional misbehavior, they also tackle the challenge of unavoidable uncertainties. However, none of them incorporate information about unavoidable uncertainties into the optimization problem, e.g., those stemming from inaccurate demand predictions. The mechanism presented in [48] mitigates unintentional uncertainties by forming coalitions of agents in such a way that prediction errors cancel each other out. In our approach, the agents pursue a similar goal when forming the hierarchical system structure, but instead of only mitigating uncertainties, the principle of homogeneous partitioning aims at forming organizations that feature a similar ratio of uncertainty to controllability. Since this promotes the organizations' ability to locally compensate for deviations, the robustness and the efficiency of the overall system increases. To reactively deal with unintentional prediction errors, Vytelingum et al. [220] introduce an online balancing mechanism that compensates for deviations between demand and supply. Given that Vytelingum et al. [220] use a Continuous Double Auction to match supply and demand, unmatched offers in the day-ahead market

can be used to implement reactive adjustments. In contrast to [220], our approach proactively identifies and deals with unavoidable uncertainties. That is, dispatchable agents schedule an appropriate amount of reserves and explicitly represent possible uncertainties in the form of scenarios in their schedules. In turn, the agents exploit this information when making reactive supply adjustments.

In the context of resource and, especially, task allocation problems, there are several approaches that propose to make use of computational trust to quantify and anticipate uncertainties originating from unavoidable and avoidable misbehavior (cf. [171] and the following references). In general, incorporating trust into the decision-making process allows the system to optimize for *expectations*, such as the expected probability of success [59]. In [148, 149], a self-organizing middleware incorporating a trust-aware load-balancing mechanism assigns important services to trustworthy nodes in order to increase the services' expected availability. Similarly, Klejnowski et al. [118] propose a Desktop Grid Computing system whose participants delegate the calculation of jobs to trustworthy agents, i.e., to members of their *Explicit Trusted Community* (see Section 6.3), to improve their expected outcome. Torrent-Fontbona et al. [212] present a trust-based approach to multi-attribute auctions in which bids consists of a price, a delivery time, and an amount of energy that is consumed if the bid is accepted. The auctioneer's goal is to accept a combination of bids that maximizes the expected utility. While these procedures are similar to the way TruCAOS employs trust values to minimize the expected remaining demand (see Section 10.4), the scheduling problem regarded in this thesis significantly differs from these task allocation problems. Consequently, these approaches do not allow for actively increasing the predictability of agents that struggle with complying with their proposals, they do not regard uncertainties by means of multiple scenarios, and they do not optimize the provision of reserves.

With regard to the reactive compensation problem considered in Chapter 12, Campbell et al. [47] investigate the role and benefit of inter-agent variation in self-organizing systems in the sense of agents with different sensitivities to changes in the demand (they refer to "sensitivity" as "error"). Similar to the reactive compensation problem, their agents aim for maintaining a task's value at a target value by making a contribution as soon as an individual tolerated deviation is exceeded. Despite a changing environment, they are able to compute how much inter-agent variation is necessary so that the system reaches the target value without any communication between the agents. This is possible because (1) the task's target value decreases by a constant value from one time step to another (which allows the agents to adjust their behavior accordingly), and (2) each agent contributes with the same constant value. In the class of open systems regarded in this thesis, heterogeneous agents are situated in a highly dynamic environment. Therefore, we cannot apply the formulas given in [47]. Instead, the agents determine useful sensitivities on the basis of anticipated uncertainties in the course of the schedule creation at runtime. Ultimately, the agents' reactions to deviations are guided by their scenario-based schedules and their individual control models.

In the context of balancing production and consumption in the power grid, Short et al. [200] present a mechanism that is based on a great quantity of devices equipped with thermal storage capacity (e.g., refrigerators). To compensate for a deviation, each device adjusts its consumption according to its state and the imbalance. To this end, the control system allows for temperatures that are linearly dependent on the deviation between production and consumption, which allows the devices, e.g., to defer cooling phases. Short et al. [200] assume that the devices are in different thermal states with respect to their temperature and thus have different sensitivities when reacting to a given deviation. As devices do not coordinate their decisions, solution quality depends on the distribution of their states. In contrast to our approach, higher level objectives, such as the minimization of costs, cannot be easily taken into account.

Similarly, van den Briel et al. [216] present an approach, called *randomized load control*, to schedule the load of a large amount of homogeneous smart appliances that representing shiftable loads. The goal is to determine the start time of the shiftable loads such that the aggregated resulting shiftable load is as close as possible to a target value. Based on the target value and its own load profile, each agent determines a probability distribution function which is used to randomly select the appliance's start time. The probability of starting the consumption in a specific time step is the higher, the greater the ratio of the target value to the appliance's shiftable load. While the approach only requires one-way communication from the utility to the smart appliances, it needs a large quantity of homogeneous appliances. Our approach proactively guides the agents reactive supply adjustments by means of

schedules that have been created on the basis of a scenario tree. Because of the agents' heterogeneity, their schedules and thus their adjustments might differ significantly.

Another approach to the balancing problem presented in [104, 105] groups small generators and consumers into pools. Within each pool, agents of smaller dynamic groups immediately cancel out minor deviations by sequentially adjusting their output. In case of significant deviations that cannot be dissolved by a dynamic group, the whole pool tries to compensate for the remaining deviation by recalculating schedules in a centralized manner. In our approach, agents do not have to recalculate schedules and decide themselves about their supply adjustments. Further, our method improves fairness with regard to the agents' ability to comply with their schedules, which is not the case in [104, 105]. Moreover, we deliberately abstain from a centralized component.

Nieße and Sonnenschein [155] introduce an approach to the balancing problem in power grids. Unlike our approach, they presume the existence of a pre-calculated set of different combinations of schedules that would re-establish the balance in case of a deviation. The main focus of their work is to choose a combination of schedules that causes a minimal impact on the power grid. In other words, their objective is to compensate for imbalances as locally as possible. Based on a weighted graph representing the power grid, they define a metric that maps output adjustments and distances to a real value. The lower the value, the more local the compensation for the deviation. However, their metric is only able to deal with one incident causing a deviation (e.g., a breakdown of a power plant) at the same time. This drastically simplifies the schedule selection since the affected agents do not have to coordinate their decisions of how to adjust the power flow in the graph. In this thesis, locality is defined on the basis of the self-organizing hierarchical system structure in which agents autonomously solve the resource allocation problem. While our approach to reactively balance supply and demand does not take account of the transmission network, it does not assume the existence of a pre-calculated set of schedules that can restore the balance, and it can deal with multiple deviations at the same time.

Résumé

In the body of literature, there are several approaches that solve problems similar to the resource allocation problem considered in this thesis. To comply with the challenges imposed by the system class, we combine techniques for proactive and reactive resource allocation. Our approach is unique in the way that we combine online stochastic optimization and self-organized hierarchical problem decomposition in order to create high-quality robust solutions in a scalable manner. Given that our approach is flanked by the auction-based algorithm TruCAOS, we cannot only create schedules for a whole tree of possible developments of the demand but also optimize the provision of reserves in systems of large size (see Section 11.5). As a result, dispatchable agents are able to deal with volatile demand that follows different behavioral patterns. In this context, the characteristics of the underlying self-organizing system structure lay the foundation for scalability and robustness (see Section 9.2). As demonstrated in our evaluations (see, e.g., Section 12.3), scenario-tree-based schedules enable the dispatchable agents to switch to the most suitable scenario at runtime. Moreover, they guide the agents' permanent reactive supply adjustments. This allows for respecting non-functional requirements considered in the process of schedule creation without having to recalculate schedules. These objectives include the cost-efficient satisfaction of the demand as well as the local compensation for deviations. TruCAOS improves the system's robustness even further by actively reducing aleatoric uncertainties (see Section 10.5). This is achieved by "moving" the dispatchable agents' supply into regions in which their behavior is especially predictable. With regard to the two dimensions of robustness outlined in Section 1.4, these characteristics improve the system's ability to remain in acceptable states and thus to maintain its functionality despite detrimental influences (first dimension), and to return into an acceptable state after a disturbance occurred that caused the system to leave the acceptance space (second dimension).

Part V

Conclusion and Outlook

Chapter 14 summarizes the findings of this thesis, its contributions, and evaluation results. Chapter 15 points to open research challenges and future directions.

Summary of Research Contributions and Evaluation Results

Summary. Summarizes the research contributions and the most important evaluation results of this thesis.

For decades, electric power systems consisted of relatively few and well-predictable power plants. Current trends and plans indicate, however, that future power systems will be characterized by new types of consumers (e.g., electric vehicles) and a vast number of distributed energy resources, including weather-dependent power plants and small potentially dispatchable generators (e.g., biogas power plants). However, the data we discussed in Section 1.3 confirm that current control schemes are already stretched to their limits and become more and more inefficient and inappropriate for balancing power production and consumption. To ensure the grid's stable operation in the future and to save expenses despite these ongoing changes, new concepts and algorithms have to be devised to control a large amount of dispatchable power plants in an uncertain and highly dynamic environment. This was our motivation for the research on an integrated approach to *robust resource allocation in open technical systems* that is based on a vision of future autonomous power systems. Our approach tackles the major challenges of this system class – scalability and uncertainty – by combining the principles of *self-organization* and *computational trust* as well as techniques from the field of *online stochastic optimization*. Extensive evaluations confirm that the concepts and algorithms presented in the previous chapters make essential and notable contributions to meet these challenges.

These evaluations were performed in an elaborate simulation environment for autonomous power systems that has been developed in the course of this dissertation in cooperation with other members of the working group. This simulation environment is built on a multi-agent system in which simulated weather conditions as well as the control models of power plants and consumers rely on real world data. As it allows for the integration of various self-organization algorithms, optimization frameworks, trust models, as well as different types of producers and consumers, it provided an excellent and realistic basis for the evaluation of the concepts and algorithms introduced in this thesis.

In the following, we summarize these concepts and algorithms as well as the most significant evaluation results:

A Trust Model to Quantify and Anticipate Uncertainties We introduced *Trust-Based Scenario Trees* (TBSTs) as an approach to probabilistic model creation. With regard to the resource allocation problem, TBSTs enable an autonomous system to deal with volatile demand that follows different behavioral patterns and whose prediction quality exhibits sequential dependencies. In such situations, relying on a single expectation of the future demand puts the system's efficiency and robustness at risk. TBSTs therefore model uncertainties by means of a set of scenarios. Each scenario represents a possible development of the demand and has a certain probability of occurrence. Compared to other

trust models, TBST's thus allow for a much more detailed representation of possible inaccuracies in demand predictions. In the context of the power management case study, we demonstrated that TBSTs are much better predictive models than a trust model based on simple trust values. On top of that, a single scenario of a TBST already reaches a similar quality of expectations as hidden Markov models. Situations in which we allowed the agents to identify and switch to the most suitable scenario at runtime confirmed the TBSTs' advantage of taking different possible developments of the demand into account. Together with techniques from the field of online stochastic optimization, TBSTs serve as the basis for the creation of robust solutions to the scheduling problem.

Specification of Necessary Characteristics of Self-Organizing Hierarchies In many multi-agent systems, a group of agents has to solve an optimization problem whose complexity is subject to the number of agents involved. If this problem can be decomposed in a hierarchical manner, hierarchical system structures are a way to obtain scalability. An example of such an optimization problem is the scheduling problem considered in this thesis. We introduced the core concepts for self-organizing hierarchical system structures that enable autonomous problem decomposition in large-scale open technical systems. Essentially, hierarchies evolve by recursively solving the *partitioning problem* in which a set of agents is partitioned into disjoint subsystems, each of which represents a smaller, tractable sub-problem of the overall scheduling problem. To steer the formation of hierarchical system structures, we introduced *partitioning constraints* that define suitable ranges for the number and the size of partitions. Moreover, we presented *homogeneous partitioning* as a new organizational paradigm that yields structures consisting of similar organizations. Such structures are especially beneficial when a system has to deal with uncertainties originating from non-dispatchable agents that cannot or should not be excluded from the system, such as weather-dependent power plants. By ensuring that each organization is equally able to cope with disturbances originating from non-dispatchable agents, homogeneous partitioning increases the system's robustness in terms of its ability to hold the balance between supply and demand. Furthermore, the complexity of solving the scheduling problem is equalized among the organizations, which is beneficial to the overall runtime performance. Throughout our evaluations, homogeneous partitioning as well as the ability to steer the degree of problem decomposition by means of the partitioning constraints proved to be beneficial to runtime performance, solution quality, and the structures' stability.

Algorithms for the Self-Organized Formation of Hierarchical System Structures Finding the optimal solution to the partitioning problem constitutes an NP-hard combinatorial optimization problem. We presented two self-organization algorithms, called *PSOPP* and *SPADA*, that solve the partitioning problem in an efficient and effective manner. While PSOPP is based on particle swarm optimization, SPADA implements a decentralized agent-based approach. In contrast to related approaches, PSOPP and SPADA solve the partitioning problem in a general manner, i.e., independently of the characteristics of a specific objective function. This allows a system to establish adequate partitionings according to different formation criteria, including homogeneous partitioning, clustering, and anticlustering. Furthermore, PSOPP and SPADA create partitionings that comply with the partitioning constraints. Our evaluations confirmed that both PSOPP and SPADA are able to create high-quality partitionings with regard to different combinations of formation criteria and parametrizations of the partitioning constraints. Of course, both algorithms have individual strengths: PSOPP outperforms SPADA in case of single objectives where its centralized and population-based approach helps in exploiting specific characteristics of the fitness landscape, such as gradients. SPADA, however, finds superior partitionings in the presence of multiple objectives. Due to their ability to abide by the partitioning constraints, PSOPP and SPADA are ideally suited for the creation of hierarchical system structures. In-depth evaluations demonstrated that our approach to self-organizing hierarchies enables a large-scale system to come to a compromise between runtime performance and solution quality. The algorithms' ability to create optimized homogeneous partitionings proved to be a key aspect to obtain stable and scalable structures that allow for high-quality solutions to the scheduling problem. Self-organizing hierarchical system structures lay the foundation for robust, efficient, and scalable resource allocation in open technical systems.

Concepts and Algorithms for Robust Resource Allocation in Self-Organizing Hierarchies

On the basis of Trust-Based Scenario Trees (TBSTs) and techniques from the field of online stochastic optimization, we defined *robust solutions* to the scheduling problem in self-organizing hierarchical system structures. In empirical evaluations, we demonstrated the advantage of TBST-based schedules that enable dispatchable agents to *locally* deal with inaccurate demand predictions. To improve the agents' ability to locally compensate for imbalances and to cope with unforeseen situations, we further extended the scheduling problem by the optimized provision of degrees of freedom, i.e., *reserves*. Because TBST-based schedules stipulate how many resources which dispatchable agent has to provide in which situation, they *proactively guide* the dispatchable agents' reactive supply adjustments. This avoids situations in which deviations between supply and demand are higher than (technically) allowed, or where their compensation is either very costly or not feasible due to the agents' inert behavior. Our evaluation confirmed that TBST-based reactive supply adjustments increase the system's efficiency in terms of costs and its ability to locally compensate for imbalances. To cope with the complexity of the extended scheduling problem, we presented an auction- and trust-based mechanism, called *TruCAOS*, that enables dispatchable agents to actively participate in the process of schedule creation. TruCAOS's excellent runtime performance allows for larger subsystems, which is advantageous with regard to the system's efficiency and stability. Our evaluations demonstrated that TruCAOS clearly outperforms a regio-central approach that is based on a state-of-the-art optimizer in terms of runtime performance, demand satisfaction, provision of reserves, and costs. Beyond that, TruCAOS further increases the system's robustness by incentivizing compliance with schedules. In particular, our evaluation showed that TruCAOS actively reduces the presence of aleatoric uncertainties originating from schedule violations.

Together, these concepts, techniques, and algorithms embody a broad and well-conceived integrated approach to self-organized and trust-based resource allocation in large-scale open technical systems.

Open Research Challenges and Future Directions

Summary. This chapter summarizes future work on the techniques, algorithms, and concepts outlined in this thesis. Furthermore, we point to complementary research areas that have to be investigated en route to self-organized resource allocation in open technical systems.

While this thesis makes important contributions for self-organized robust resource allocation in open technical systems, it leaves room for future investigations by revealing relevant research questions. In the course of our work, we identified a number of possibilities for advancing the proposed concepts and algorithms. As, of course, this thesis cannot address all relevant issues of the regarded system class; also further aspects have to be considered on the way to a practical adoption of techniques based on self-organization and computational trust.

With regard to the concept of Trust-Based Scenarios (TBSTs), we pointed out that a combination with other approaches to predictive modeling, such as hidden Markov models or artificial neural networks, could further improve the quality of created probabilistic models. A hybrid approach could use such techniques together with the recent measurements of prediction quality to identify the most appropriate subtree in a TBST that includes look-back behavior. While we only considered the creation of TBSTs on the basis of observations made at runtime, it is worthwhile to investigate the inclusion of historical data. Because this is likely to increase the number of scenarios, techniques for *scenario reduction* (cf. [78]) could be applied in order to create high-quality TBSTs consisting of a relatively low number of scenarios. Maintaining a relatively low number of scenarios is of particular interest given that the complexity of the scheduling problem also depends on the number of considered scenarios.

As for the partitioning algorithm PSOPP, we mentioned that its extension to multiset partitioning problems could be useful to increase the algorithms performance in situations in which a system contains many components with identical properties. Because this extension complicates the application of the operators PSOPP's particles use to approach other candidate solutions, the influence on PSOPP's performance is hard to predict. To improve PSOPP's performance in multi-objective contexts, a combination with an evolutionary algorithm could reduce the chance that PSOPP's particles get trapped in local optima (cf. [3]). In such a situation, it could also be beneficial to allow PSOPP to make use of solutions lying on the pareto frontier. Regarding SPADA, it would be advantageous to improve its ability to create new partitions in case of relatively high thresholds for the minimum size of partitions. In this context, it would be interesting to extend the optimization of regional partitionings by key aspects of PSOPP's move operations.

Recall that our approach to self-organizing hierarchies is based on control actions that either dissolve intermediaries, introduce new intermediaries, or re-partition an intermediary's neighborhood. Essentially, these *regional* control actions only affect a part of a specific layer in a hierarchy. While this characteristic is beneficial for the efficient creation of the structure, it can hinder re-establishing a proper system

structure. Imagine, for example, a hierarchy of height > 2 in which it would be necessary to exchange an agent residing in the bottom left of the hierarchy with an agent residing in the bottom right. In this situation, it is likely to be insufficient to re-partition a partitioning at a higher hierarchy level that does not directly include these agents, or to re-partition the independent neighborhoods of the intermediaries controlling these agents. The challenge is to identify such situations and to trigger a reorganization that incorporates the relevant parts of the system structure, e.g., by carrying out an appropriate *series* of the above-mentioned control actions. Similar methods could also be useful to deal with reorganizations needed after agents entered and left the system. In such situations, a diagnosis of the current state of affected subsystems could guide a re-partitioning performed by PSOPP or SPADA. This would allow for re-establishing an appropriate structure by means of a low number of modifications. If agents enter and leave the system frequently, such techniques could also be applied in ongoing reorganizations.

The evaluation of self-organizing hierarchies showed that their efficiency and stability depends heavily on the parametrization of the introduction, the dissolution, and the partitioning constraints. So far, we used a parametrization that did not change during a simulation run. Because it is not feasible to specify a set of parameters suitable for all situations that occur during a system's lifetime at design time, the agents should be able to identify these parameters by themselves and adapt them according to the current circumstances. This is a challenging task because the complexity of the scheduling problem depends on various dynamic factors, such as the shape of TBSTs (i.e., degree of uncertainty), the subsystem's composition, the agents' current state, and the predicted demand. Furthermore, overall runtime performance depends on the width and the height of the hierarchy. These properties also influence the structure's stability and solution quality in terms of demand satisfaction and costs. A possible candidate for identifying suitable parametrizations at runtime could be *Bayesian optimization* of hyperparameters, which has been shown to outperform grid search and random search.

As for TruCAOS, it would be interesting to investigate if the system's efficiency can be further increased by enabling dispatchable agents to consider individual preferences when creating proposals. For instance, a base load power plant could propose a schedule in which the plant's output is not subject to changes in the residual load. Because it would be insufficient to regard such preferences exclusively at the level of agents representing physical devices, intermediaries would have to derive an abstracted preference structure from its subordinates' preference structures. *Constraint relationships* [188] or, more generally, partial valuation structures constitute promising candidates for this endeavor. In practical applications of TruCAOS, it would also be of interest to ensure a fair allocation of resources among the participants [164]. Given that schedules address multiple demand scenarios and also incorporate the provision of reserves, appropriate fairness metrics have to be devised that comply with the different roles the agents take on according to their preferences and capabilities.

With regard to the TBST-based schedule creation in hierarchical systems, the system's robustness could be improved by deriving TBSTs that represent uncertainties stemming from different levels in the hierarchy. While we actually want to avoid the top-down propagation of uncertainties, an intermediary has sometimes no choice but to engage a subordinate intermediary to compensate for a part of its local uncertainties. In such situations, the subordinate intermediary should derive a combined TBST from the scenarios contained in its schedule and its local deviation tree. In our current approach, the subordinate intermediary only combines the anticipated scenario of its schedule with its local deviation tree. To obtain a meaningful combined TBST, i.e., an adequate representation of the combined uncertainties, the subordinate intermediary has to identify correlations between the prediction errors of its own local demand and those of the demand at higher system levels. Again, techniques from the field of scenario reduction come into play.

To increase the system's runtime performance without impairing its robustness and efficiency, an interesting idea is the on-demand recalculation of schedules. That is, to delay the recalculation of schedules until it is observed that the present schedules might not adequately guide the dispatchable agents' reactive supply adjustments in future. The use of an adaptive scheduling window that contains a flexible number of time steps with a situation-specific temporal distance might further rise the system's robustness and efficiency.

En route to the practical adoption of self-organization algorithms, further challenges have to be tackled. An open research question is how to systematically assure the quality of self-organizing systems

within a software development process [63]. An established measure for assuring the functional quality of a system is testing. Here, the key challenge is to cope with error masking, interleaved feedback loops, the oracle problem, and huge state spaces [65]. A self-organizing system's non-deterministic behavior and its dynamic environment complicate this undertaking. Because the overall system's performance is subject to the performance of the employed self-organization algorithms, one might further be interested in choosing the best-performing algorithms for a specific task. Due to the self-organizing systems' characteristics, new metrics have to be devised that assess the performance of these algorithms with regard to different criteria, such as runtime performance or robustness [64]. Such metrics cannot only be used to identify the most suitable algorithm and an appropriate parametrization at design time, but also to switch to the best algorithm and configuration at runtime. The latter is especially beneficial because the environmental conditions that influence the algorithms' performance are partially unpredictable at design time. For instance, such metrics could be used to switch between PSOPP and SPADA or between TruCAOS and the regio-central approach at runtime; different parts of the hierarchy might even make individual choices.

With regard to autonomous power systems, additional challenges have to be addressed when taking further power grid infrastructures into account. The transmission network, for instance, imposes supplementary constraints, such as line capacities and voltage bounds [35]. However, ensuring adequate power flows through the system requires solving additional optimization problems [154]. While we focused on power generation in this thesis, the transmission network's stability, in general, could substantially benefit from incorporating dispatchable power consumers into our concept of AVPPs, as well. Taking these further considerations into account, the common ideas of the concepts and algorithms devised in this thesis could be helpful to deal with the complexity of these problems and the uncertainties the system is exposed to. The constraints of the transmission network not only have to be regarded when creating schedules, but also during the formation of the hierarchical structure of AVPPs. As we explained in Section 9.1, our approach to self-organizing hierarchies allows for self-organization on top of partially predefined system structures, such as the transmission network. Predefined entities of the transmission network, e.g., in the form of groups of subnetworks that allow for adequate power flows, could be identified by means of graph partitioning algorithms, such as the one proposed by Rahimian et al. [169]. But even within such predefined entities, one might have to take account of the transmission network's characteristics when forming AVPPs. For instance, one might prefer to group power plants and consumers residing in the same low-voltage network. Such additional objectives (e.g., expressed in terms of constraint relationships) have to be combined with the original objective function of the partitioning problem, e.g., by a lexicographical optimization or other multi-objective techniques from the literature.

Actually, new technology, such as combined heat and power plants or power to gas, connects different supply systems with each other. For instance, combined heat and power plants generate electricity that flows into the power grid while, at the same time, producing heat that is fed into district heating systems. Yet, combined heat and power plants introduce uncertainties because they might unexpectedly increase their output if more heat is requested than was predicted. As for power to gas, electricity is converted into gas that can be fueled into gas pipeline systems that provide a huge amount of storage capacity. The stored gas can be used, for instance, to fire a gas heating or, at a future point in time, to run a combined heat and power plant. Power to gas can therefore compensate for a surplus of electricity. To reduce uncertainties and take advantage of these new technologies, it is only logic to not only focus on one supply system separately, but to treat them as the *interwoven system* [211] they are. However, interwoven systems impose completely new challenges as not only the provision of a single good, such as power, has to be optimized but of several goods at the same time. Mutual influences between the supply systems and their conflicting goals, as well as the interwoven system's huge size complicate this task. The concepts and algorithms we developed in this thesis lie an excellent foundation for such considerations.

As for future power systems, our evaluation results confirm the applicability, suitability, and necessity of the presented concepts and algorithms. In particular, a combination of our techniques with the outcomes of other projects, all investigating different aspects of these systems, would promise the most robust, scalable, and flexible solution. We are therefore convinced that this thesis makes a significant contribution to pave the way to future self-organizing autonomous power systems.

List of Figures

1.1	The residual load is the difference between the overall load of non-dispatchable consumers and the overall output of non-dispatchable power plants. It has to be satisfied by the dispatchable power plants and the dispatchable consumers. Note that the residual load is negative if the non-dispatchable output exceeds the non-dispatchable load.	5
1.2	Schedules stipulate the dispatchable prosumers' output on the basis of residual load predictions (cf. the numbers in the boxes) for a specific time span in advance. Here, schedules cover a time span of $H = 45$ min and are created with a resolution of $\Delta\tau = 15$ min. Hence, each schedule comprises $N = 3$ time steps. At 10:00, for instance, schedules are created for the scheduling window $\mathcal{W} = \{10:15, 10:30, 10:45\}$. As predictions become more accurate as a future point in time approaches, schedules are periodically revised (here, every 15 min). Each schedule creation is depicted in a different color. . .	7
1.3	Once a violation of the corridor of correct behavior is detected, the system triggers a reorganization that re-establishes compliance with the invariant (see Figure 1.3a). Refining the corridor by means of a target space allows the system to preserve its efficiency by triggering a reorganization before it leaves the acceptance space (cf. the second reorganization in Figure 1.3b). In this case, reorganizations aim at bringing the system back into an optimal state.	10
2.1	Hierarchical system structure of a future autonomous and decentralized power system: Power plants are structured into systems of systems represented by AVPPs that act as intermediaries to decrease the complexity of control and scheduling. AVPPs can be part of other AVPPs. The left child of the top-level AVPP, for instance, controls a coal power plant, a wind turbine, and two subordinate AVPPs. The leftmost AVPP, on the other hand, controls only physical power plants; from left to right: a storage battery as well as a solar, a biomass, a hydro, and a gas turbine power plant.	16
4.1	The life-cycle of trust values derived from experiences (adapted from [204]).	27
4.2	Example of an inaccurate demand prediction: The dotted vertical lines indicate the time steps $t_i = t_{\text{at}} + i \cdot \Delta\tau$ that constitute the scheduling window \mathcal{W} in time step t_{at} . As illustrated in Figure 4.2a, uncertainties manifest as deviations between the actual $\langle 35, 35, 30 \rangle$ and the predicted demand $\langle 85, 90, 90 \rangle$. With regard to t_1 , for instance, the intermediary gains an atomic experience $E_{t_{\text{at}}}[t_1] = (35, 85, t_{\text{at}}, t_1)$ that captures the deviation $\delta(E_{t_{\text{at}}}[t_1]) = 35 - 85 = -50$. The corresponding sequential experience $E_{t_{\text{at}}}$ records the sequence of deviations $\langle -50, -55, -60 \rangle$. Figure 4.2b shows that the influence of inaccurate predictions can be mitigated by relying on expectations instead of the prediction itself. An intermediary derives an agent's expected demand from its demand prediction and the trust value reflecting its expected deviation.	28

- 4.3 Four demand predictions made for $N = 3$ consecutive time steps and a schedule resolution $\Delta\tau = 15$ min at four different points in time $t_{\text{at}} \in \{10:00, 10:15, 10:30, 10:45\}$. The figure illustrates the relation between a demand prediction $\hat{D}_{t_{\text{at}}}^{\mathcal{W}}[t_{\text{for}}]$ for a specific time step t_{for} and the temporal distance $\Delta h = t_{\text{for}} - t_{\text{at}}$ (in this example, $\Delta h \in \{15 \text{ min}, 30 \text{ min}, 45 \text{ min}\}$). As indicated at 10:45, the corresponding intermediary gains $N = 3$ experiences in each time step. 31
- 5.1 The creation of a *deviation tree* is a four-step process: In step 1, a number of sequential experiences is normalized (here we use an interval $[-1, 1]$ and a normalization factor of 100). Afterwards, the normalized sequential experiences are classified by means of a grid of bins while maintaining the information about sequential dependencies. A set of classified experiences that results in the same sequence of bins constitutes a *deviation scenario*, as is the case with Experiences 1 and 2 (cf. step 2). In step 3, we regard the set of deviation scenarios as a tree whose branches result from common prefixes of the sequences of bins (each path from the root r to a leaf represents a deviation scenario). The numbers in the tree's nodes indicate how often a bin is reached on a specific path starting at the root. By using relative instead of absolute frequencies, we obtain, for each inner node, a relative frequency distribution of its child. The result is a deviation tree with conditional probabilities as shown at the tree's transitions in step 4. A scenario's probability of occurrence is equal to that of the corresponding leaf. For example, the probability of the uppermost deviation scenario $\langle [-0.56, -0.33], [-0.6, -0.2], [-1, -0.33] \rangle$ in step 4 is ≈ 0.34 38
- 5.2 An illustration of a deviation tree (lower part of the nodes) and a demand tree (upper part of the nodes) of an intermediary λ obtained from a demand prediction made for the scheduling window $\mathcal{W} = \{t_{\text{now}} + \Delta\tau, \dots, t_{\text{now}} + 4 \cdot \Delta\tau\}$. Each path from the root r to a leaf is a TBS. As for the demand tree, every node (except for the root r) represents an expected demand in the corresponding future time step. The expected demand is calculated by adding a TBS's expected deviations (lower part of the node) to the predicted demand $\langle 10, 14, 13, 11 \rangle$. The values at the TBST's edges indicate the conditional probabilities that the demand changes from one value to another. A TBS's probability of occurrence is equal to that of the corresponding leaf. The highlighted path indicates the *anticipated scenario*, which denotes the most likely development of the demand. 40
- 5.3 Mean deviation of the expected residual load (unmodified residual load prediction, trust value, TBSs) from the actual residual load. Figure 5.3a depicts this deviation over time. Figure 5.3b shows the accuracy of the expected residual load for the 32 time slots. The curve "Unmodified Prediction" in Figure 5.3b demonstrates that the prediction error increased with the prediction horizon, i.e., the time slot a prediction was made for. . . 44
- 5.4 Development of the absolute deviation of the expected residual load from the actual residual load for a single AVPP and the first time slot over the fifth simulated day. TBSTs and HMMs substantially reduce the absolute deviation. "Best Node" indicates the deviation between expected and actual residual load when allowing the AVPP to switch to the TBSTs' node that is closest to the actual prediction error at runtime. . . 46

6.1	In a centralized system, a central authority has control over all other agents in the system and calculates schedules for all dispatchable agents (see Figure 6.1a). By creating subsystems that are represented by intermediaries, we obtain a flat system structure of height 2 that decomposes the scheduling problem into several sub-problems (see Figure 6.1b). The top-level intermediary (cf. the black node) controls all subordinate intermediaries (cf. gray nodes) which, in turn, control the agents representing physical devices (cf. white nodes). A hierarchical system structure allows the system to scale with an arbitrary number of agents by introducing new intermediaries where needed (see Figure 6.1c). Note that the hierarchy's leaves do not have to have the same depth. In a hierarchy, intermediaries control subordinate intermediaries and agents representing physical devices.	52
6.2	Temporal abstraction for an intermediary consisting of three suppliers a, b, c given their current states shown in time step t_{now} . White boxes indicate general boundaries, gray areas represent the temporal boundaries at time step $t \in \{t_{\text{now}}, \dots, t_{\text{now}} + 3\}$. Supplier a needs two time steps to start up and is then available at its minimal output. Adapted from [187].	55
6.3	Sampling abstraction is concerned with selecting informative points of an unknown but computable function $f(x)$ (bold solid line). Input and output represent variables of a collective, e.g., <i>production</i> to <i>costs</i> . An approximation $a(x)$ (solid line) of $f(x)$ is iteratively improved at its most uncertain point to yield a better approximation $a'(x)$ (dashed line). These points are provided by a regression model (dotted line). Adapted from [17].	56
6.4	Comparison of the time needed to create schedules for different numbers of power plants. "Hierarchical Sequential" corresponds to the time needed to calculate all schedules sequentially, whereas "Hierarchical Parallel" corresponds to the maximum sequential scheduling time. Results are averaged over 2400 data points from 50 drawn sets of plants each considering 48 time steps. Based on data provided in [187].	56
6.5	By partitioning the participants of a centralized system (see Figure 6.5a), we obtain a flat system structure (see Figure 6.5b). Each created partition embodies a subsystem that is represented by an intermediary (shown in gray). The hierarchical system structure depicted in Figure 6.5c results from recursively partitioning the members of the subsystem in the center of the flat system structure. Again, each created partition is represented by an intermediary (shown as dashed nodes). In both cases, the rectangle indicates the set of agents that participates in the partitioning problem.	58
6.6	Reorganization of a flat system structure: The initiating intermediary is marked in gray. The agents \mathfrak{A} participating in the re-partitioning are highlighted by the rectangle. After the reorganization, the two intermediaries " \ominus " standing for the former partitioning \mathcal{P} are replaced by the three new intermediaries " \oplus " representing the partitions of the new partitioning \mathcal{P}'	67
6.7	The single steps performed by a controller to reorganize a partitioning. The step "Make Partitioning Compliant with Constraints" is only needed if agents entered or left the system in such a way that the partitioning constraints are violated.	68

7.1	Motion of a single particle in a two-dimensional search space: In the original definition of PSO, the particle moves through a continuous search space (see Figure 7.1a). Its new position $\mathbf{x}_i(s+1)$ is determined by its updated velocity $\mathbf{v}_i(s+1)$, which is a linear combination of the particle's current velocity $\mathbf{v}_i(s)$, the vector from its current position $\mathbf{x}_i(s)$ to its best found solution \mathcal{B}_i , and the vector from $\mathbf{x}_i(s)$ to its best found solution $\mathcal{B}_{\mathcal{N}_i}$ in its neighborhood. Regarding JPSO, the particle moves through a discrete search space (see Figure 7.1b). Its new position results from either making a random move or from approaching \mathcal{B}_i , $\mathcal{B}_{\mathcal{N}_i}$, or the global best found solution \mathcal{B} . In case of an approach, the particle's similarity to \mathcal{B}_i , $\mathcal{B}_{\mathcal{N}_i}$, or \mathcal{B} increases so that its new position is somewhere within the corresponding circle depicted in Figure 7.1b. Note that $\mathbf{x}_i(s)$ lies on the circles' circumference.	82
7.2	Actions performed by a particle in each iteration.	84
7.3	Parameter search: Fitness obtained by PSOPP for the objectives C, AC, HPm, and HPs when partitioning 1000 elements and 4 particles with different combinations of c_{rdm} , $c_{\mathcal{B}_i}$, $c_{\mathcal{B}}$. Recall that $c_{\mathcal{B}} = 1 - c_{rdm} - c_{\mathcal{B}_i}$. Results are averaged over 100 runs per parameter combination. Note that the displayed fitness ranges vary from one objective to another.	92
7.4	PSOPP's and RDM's convergence with regard to raw values for $n = 2000$, $\#P = 4$, and a time limit of 60s in case of objective C (average of 500 runs). PSOPP achieves an average fitness of 0.89, 0.98, and 1.00 after 10s, 30s, and 60s, respectively. In comparison, RDM obtains 0.54, 0.55, and 0.55.	94
8.1	SPADA's iterative optimization in form of optimization cycles. In every optimization cycle, each of the n agents participating in the PP performs an optimization step in which it tries to increase the fitness of its regional partitioning. The agents can perform their optimization steps either sequentially or in parallel. $OS_{c,j}$ denotes the c -th optimization step of agent a_j	100
8.2	Figure 8.2a shows an acquaintances graph for a set $\mathfrak{A} = \{a_1, \dots, a_6\}$ of six agents, each of which is represented by a node. There are two acquaintances per agent, symbolized by arcs. Agent a_6 , for instance, is acquainted with a_2 and a_3 . Figure 8.2b depicts a partitioning of \mathfrak{A} consisting of the three color-coded partitions $\{a_1, a_3, a_6\}$, $\{a_2, a_4\}$, and $\{a_5\}$. Their leaders a_1 , a_4 , and a_5 are indicated by double circles. Light thin and dark bold arcs denote unmarked and marked edges, respectively.	102
8.3	An acquaintances graph for a set $\mathfrak{A} = \{a_1, \dots, a_{16}\}$ of sixteen agents, each having two acquaintances. The partitioning is composed of five partitions $\{K, \dots, O\}$, represented by the five larger circles. The partitions' leaders are indicated by double circles. Light thin and dark bold arcs denote unmarked and marked edges, respectively.	103
8.4	Actions leaders perform during their optimization step in each optimization cycle. . . .	104
8.5	An alternative view onto the example given in Figure 8.3: The regional partitioning $\mathcal{R} = \{K, L, N, O\}$ of partition N is highlighted by the large dark circles. Marked edges are symbolized by the bold solid arcs. All other arcs denote unmarked edges. The unmarked edges of N are accentuated by the dashed arcs. A partition's regional partitioning is defined by the heads of its unmarked edges and always contains the partition itself. Therefore, N would also be in \mathcal{R} if the unmarked edge (a_{13}, a_{12}) did not exist.	106
8.6	Overview of the four stages to correct candidate transactions.	117
8.7	The changes made by a leader $l(K_{\text{from}})$ to detach an agent a_i from the arborescence representing its partition: Figure 8.7a depicts the removal of an internal node that is not the root. Figure 8.7b shows the removal of the root, i.e., the leader $l(K_{\text{from}})$. In this case, $l(K_{\text{from}})$ appoints one of its children to be the new leader $l^*(K_{\text{from}})$. In both cases, the root of the resulting arborescence is indicated by a double circle. Solid bold arcs symbolize existing marked edges, dashed bold arcs symbolize new marked edges, and dashed non-bold arcs symbolize marked edges that are converted into unmarked edges.	119

8.8	Unrestricted partitionings: Development of the fitness values obtained by SPADA, PSOPP, and RDM for objectives C and HPm over a time frame of 10 s. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.	124
8.9	Restricted size of partitions: Development of the fitness values obtained by SPADA, PSOPP, and RDM for objective C, 1000 agents, and different restrictions of the minimum and the maximum partition size over a time frame of 10 s. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.	126
8.10	Comparison of the local to the centralized correction method: Development of the fitness values (see Figure 8.10a) and the corresponding standard deviations (see Figure 8.10b) obtained by SPADA using either the local correction of candidate transactions or the centralized correction of partitionings. Although we limited SPADA's runtime to 15 s, we show data for 16 s because the centralized approach has to perform a final correction before returning the solution. The fitness values are averages over 100 runs. In all runs, we used objective C in combination with problem structure PS-09 consisting of 1000 agents ($s_{min} = 2$, $s_{max} = 20$, $n_{min} = 270$, and $n_{max} = 280$).	128
8.11	Restricted size and number of partitions: Development of the fitness values obtained by SPADA, PSOPP, and RDM for objective C, 1000 agents, and different restrictions of the number-of-partitions constraints over a time frame of 15 s. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.	129
8.12	First multi-objective evaluation scenario: Development of the fitness values (see Figure 8.12a) and the corresponding standard deviations (see Figure 8.12b) attained by SPADA, PSOPP, and RDM over a time frame of 120 s. Results are obtained for the problem structure PS-10 (1000 agents, $s_{min} = 20$, $s_{max} = 50$, $n_{min} = 20$, and $n_{max} = 40$) in a multi-objective setting combining one objective for C, one objective for HPm, and four objectives for HPs. The fitness values are averages over 100 runs. In case of PSOPP, the data is based on the global best found solution.	130
8.13	Second multi-objective evaluation scenario: Development of the fitness values attained by SPADA, PSOPP, and RDM over a time frame of 60 s. Results are obtained for the problem structure PS-10 (1000 agents, $s_{min} = 20$, $s_{max} = 50$, $n_{min} = 20$, and $n_{max} = 40$) in a multi-objective setting combining one objective for AC, one objective for HPm, and four objectives for HPs. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.	132
8.14	Third multi-objective evaluation scenario: Development of the fitness values attained by SPADA, PSOPP, and RDM over a time frame of 400 s. Results are obtained for the problem structure PS-11 (4000 agents, $s_{min} = 40$, $s_{max} = 100$, $n_{min} = 40$, and $n_{max} = 80$) in a multi-objective setting combining one objective for AC, one objective for HPm, and four objectives for HPs. All fitness values are averaged over 100 runs. In case of PSOPP, the data reflects the development of the fitness of the global best found solution.	133
9.1	The Hierarchical Control Loop as performed by an intermediary λ : The intermediary monitors the violation of specific constraints, depicted as guards on the transitions. In case of a violation, it triggers an associated control action that either introduces new intermediaries, dissolves the intermediary, or reorganizes the partitioning its children belong to. In the latter case, the <i>neighborhood</i> denotes the set of agents to be re-partitioned. It consists of the intermediary's children and nephews. Adapted from [207].	137
9.2	Dissolution of an intermediary: The initiating agent is marked in gray. Before its dissolution, it transfers its children " \odot " to its parent " \otimes ".	138

9.3	Introduction of new intermediaries: The initiating intermediary is marked in gray. Its children – highlighted by the rectangle – participate in a partitioning problem. Each created partition is represented by a new intermediary “ \oplus ” shown in black.	139
9.4	Reorganization of an intermediary’s neighborhood: The initiating intermediary is marked in gray. Its neighborhood is highlighted by the rectangle. The dashed intermediary represents a predefined organization and cannot be dissolved, which is why its children are excluded from the neighborhood. After the reorganization, the intermediaries “ \ominus ” are replaced by new intermediaries “ \oplus ” shown in black.	140
9.5	Mean lifetime of AVPPs in time steps for different combinations of maximal scheduling times, the size of partitions, the employed objective function, and the type of dissolution and introduction constraints. Where not stated otherwise (see “No-MaxSPAN” and “Min. Fitness”), AVPPs employed MaxSPAN constraints as dissolution and introduction constraints, and agents aimed for homogeneous partitionings. The AVPPs’ lifetime is improved by using higher thresholds for the maximal scheduling time, increasing the size of partitions, and establishing homogeneous partitionings. The maximal possible lifetime amounts to 96 time steps.	145
9.6	Development of the mean number of reorganizations performed over time. The agents’ ability to self-organize is enabled after time step 16. For all configurations, results are averaged over 100 runs. The lower the number of reorganizations, the more stable the system structure.	145
9.7	Development of the mean height of the hierarchy (see Figure 9.7a) and the mean number of AVPPs (see Figure 9.7b) over time. The agents’ ability to self-organize is enabled after time step 16. For all configurations, results are averaged over 100 runs. The initially centralized system structure has a height of one and consists of a single AVPP.	146
9.8	Development of the maximum sequential scheduling time for a centralized system without the capability to self-organize, compared to a configuration in which the agents’ ability to self-organize is enabled after the 16-th time step. For both configurations, results are averaged over 100 runs. The development of the scheduling times of the centralized system reveals that the complexity of solving the scheduling problem varies with the time of day.	146
9.9	Comparison of the development of the maximum sequential scheduling time for different configurations in which the agents’ ability to self-organize is enabled after the 16-th time step. For all configurations, results are averaged over 100 runs.	147
9.10	Comparison of the development of the mean scheduling time per AVPP for different configurations in which the agents’ ability to self-organize is enabled after the 16-th time step. For all configurations, results are averaged over 100 runs. Regarding the average of the mean scheduling time per AVPP over all time steps, we obtain 56.65 ms ($\sigma = 14.78$ ms) for “Large Partitions (Max. Sched. Time 60 ms)”, 66.13 ms ($\sigma = 11.41$ ms) for “Large Partitions (Max. Sched. Time 60 ms, Min. Fitness)”, 137.16 ms ($\sigma = 23.22$ ms) for “Large Partitions (Max. Sched. Time 600 ms)”, and 140.17 ms ($\sigma = 15.75$ ms) for “Large Partitions (Max. Sched. Time 600 ms, Min. Fitness)”.	148
10.1	TruCAOS’s basic procedure.	156
10.2	The steps performed for the activity “Identify and Accept Winner Proposals”.	160

10.3	A biogas power plant's predictability dependent on its output: We assume that the power plant's gas storage is constantly filled at a rate r . Usually, such a power plant can burn gas at a higher rate than r , which allows the operator to regulate the pressure in the storage by adjusting the power plant's output. If the storage's capacity reaches an upper bound, the power plant's output has to be increased in order to decrease the pressure in its gas storage. Note that this measure might violate the power plant's schedule (i.e., it misbehaves). As a consequence, the power plant features a medium predictability in situations in which it should be turned off. On the other hand, if the power plant runs at (almost) full capacity for a longer period of time, the gas pressure can reach a lower bound. In such a situation, the power plant might violate its schedule since it has to decrease its output. This causes a medium to low predictability at higher output levels. To reduce aleatoric uncertainties originating from this power plant, its scheduled output should be shifted into regions of higher predictability.	163
10.4	The mean predicted \hat{D} , expected \bar{D} , and actual D residual load. The mean absolute deviation $\Delta(D, \hat{D})$ between D and \hat{D} reflects the predefined randomly generated prediction error. As for E-t, the mean absolute deviation $\Delta(D, \bar{D})$ between D and \bar{D} mirrors the accuracy of \bar{D} , which is provided by the presented trust metrics.	170
10.5	The promised \hat{S}_{MPP} and actual S_{MPP} supply of the MPP in E-nt. The mean absolute deviation $\Delta(S_{\text{MPP}}, \hat{S}_{\text{MPP}})$ between S_{MPP} and \hat{S}_{MPP} mirrors the MPP's varying and contribution-specific behavior.	170
10.6	The promised \hat{S}_{MPP} , expected \bar{S}_{MPP} , and actual S_{MPP} supply of the MPP in E-t. As in Figure 10.5, the mean absolute deviation $\Delta(S_{\text{MPP}}, \hat{S}_{\text{MPP}})$ between S_{MPP} and \hat{S}_{MPP} mirrors the MPP's varying behavior. The fact that the mean absolute deviation $\Delta(S_{\text{MPP}}, \bar{S}_{\text{MPP}})$ between S_{MPP} and \bar{S}_{MPP} is smaller than $\Delta(S_{\text{MPP}}, \hat{S}_{\text{MPP}})$ confirms that TruCAOS's trust-based decisions mitigate epistemic uncertainties.	170
10.7	The mean absolute deviation $\Delta(D, S_V)$ between the actual residual load D and the total actual supply S_V in E-nt compared to E-t. TruCAOS's trust-based decisions significantly decrease $\Delta(D, S_V)$	170
10.8	The number of actual contributions of the MPP's twin (see Figure 10.8a) and the number of expected contributions of the MPP (see Figure 10.8b) for time steps $t \in [192, 384]$. The width of the intervals corresponds to 10 kW. The MPP's sweet spot of high predictability can be found at 225 kW. The MPP's twin always complies with its scheduled supply and is thus very predictable.	171
11.1	A fitness landscape illustrating the idea of robust solutions (according to [42]): A conventional optimization method searches for the optimal solution that might only be feasible for a specific scenario (e.g., it finds the least expensive schedules for a residual load of 10 MW). In case of disturbances, the solution's actual fitness might turn out to be much lower, though (e.g., assume that the actual residual load is 12 MW so that a costly peaking power plant has to be ramped up to satisfy the additional demand). A robust optimization method incorporates uncertainties in the form of multiple scenarios. Robustness is achieved by, for instance, searching for a plateau of good fitness values that yields a higher expected fitness than the solution of the conventional optimization method (e.g., as a residual load of either 10 MW or 12 MW is expected, it schedules the power plants in such a way that the residual load can be satisfied with inexpensive power plants in both scenarios).	179

- 11.2 Different ways to consider and deal with the scenarios contained in a scenario tree (scenarios are color-coded): While multi-stage problems take account of the tree structure (see Figure 11.2a), two-stage problems consider the same scenarios in the form of a fan of unrelated scenarios (see Figure 11.2b). Figure 11.2c shows a variant of two-stage problems presented in [193]. This variant abolishes recourse actions for the next relevant time step (cf. the root's child) by representing the uncertainties by a single expected value. The expected value is the average of the root's children in Figure 11.2a. Evidently, regarding an expected value does not allow for encoding strategies for how to react to different situations. 180
- 11.3 The power plants use TBST-based schedules as a blueprint for how much power to provide in which situation. For each node, the sum of the scheduled output (upper half of the nodes) of the base load and the peaking power plant matches the corresponding expected residual load (lower half of the nodes). For instance, the base load power plant should provide 60 MW and the peaking power plant 20 MW in case of a residual load of 80 MW in t_1 (here, $t_i = t_{\text{now}} + i \cdot \Delta\tau$ with $i = \{1, 2, 3\}$). While the base load power plant should provide an output that is more or less constant and independent of changes in the residual load, the peaking power plant – whose maximum output is 20 MW – should make adjustments in accordance with residual load changes. 182
- 11.4 An example of a demand tree $D_{\lambda}^{\mathcal{T},dis}$ specifying the residual load the AVPP λ has to distribute (lower part of the nodes) and a corresponding schedule $S_a^{\mathcal{T}}$ for a peaking power plant $a \in \mathcal{V}_{\lambda}$ (upper part of the nodes): Here, we regard a scheduling window $\mathcal{W} = \{t_{\text{now}} + \Delta\tau, \dots, t_{\text{now}} + 3 \cdot \Delta\tau\}$. As for the demand tree $D_{\lambda}^{\mathcal{T},dis}$, each path from the root r to a leaf is a TBS that embodies a possible development of the residual load λ is accountable for. Except for r , every node represents an expected residual load in a future time step $t \in \mathcal{W}$ (all values in MW). With respect to the schedule $S_a^{\mathcal{T}}$, each node specifies the supply the peaking power plant should provide in case the actual residual load is equal to the corresponding expected residual load. For instance, if the actual residual load equals 80 MW in time step $t_{\text{now}} + 2 \cdot \Delta\tau$, the peaking power plant should provide an output of 20 MW. If the residual load is only 60 MW, the peaking power plant should be turned off. The values at the TBST's edges indicate the conditional probabilities that the residual load changes from one value to another. As stated in Section 5.2, we define the probability $p(n)$ that a node n occurs as follows: $p(r) = 1$, $n \neq r \rightarrow p(n) = p(n | f(n)) \cdot p(f(n))$, where the function f returns the parent of a node n . Hence, a TBS's probability of occurrence is equal to that of the corresponding leaf. The highlighted path indicates the *anticipated scenario*, i.e., the most likely development of the residual load. 183
- 11.5 Top-down creation of TBST-based schedules according to the meso-level approach. The bold arcs in scenario trees highlight the anticipated scenario. The steps (1b), (3), (4a), and (4b) are performed by the top-level intermediary Λ , whereas the steps (1a), (2), (5), (6a), and (6b) are performed by the intermediary λ_2 . The value -1 between the steps (1a) and (2) represents λ_2 's local demand prediction. The local demand prediction of Λ is composed of λ_1 's anticipated local demand of 4 (shown between the steps (1b) and (3)) and λ_2 's anticipated local demand of -3 . The latter is extracted from λ_2 's local demand tree in step (2). The actions performed by the intermediaries λ_1 , λ_3 , and λ_4 are equivalent to those performed by λ_2 . For the sake of clarity, we do not depict these actions in the figure. We explain this figure step by step in the course of the examples given in this section. 186

11.6	Two illustrations of the <i>available positive reserves</i> $R_a^{avl,+}[n]$ a dispatchable agent $a \in \mathcal{V}$ representing a physical device can provide between two nodes $f(n)$ and $n \in \{n_1, n_2\}$ (available reserves are indicated by the gray areas): n_1 stands for the case in which the agent a should increase its supply from $t - \Delta\tau$ to t , whereas n_2 represents the case in which a should decrease its supply. The amount of $R_a^{avl,+}[n]$ is subject to a 's scheduled supply $S_a^x[f(n)]$ and $S_a^x[n]$ as well as its control model. The latter captures a 's inertia represented by the function \vec{S}_a^{\max} as well as its minimal S_a^{\min} and maximal S_a^{\max} supply, among others. With regard to its scheduled supply $S_a^x[f(n)]$ and $S_a^x[n]$ (we assume a linear change of a 's supply between $f(n)$ and n), $R_a^{avl,+}[n]$ denotes, e.g., the <i>minimum</i> (in case of a <i>pessimistic</i> approach) of the maximal additional supply that a can mobilize within the single time steps between $f(n)$ and n . In this context, $C_a^{\max,+}[n]$ is the maximal supply a can provide between $f(n)$ and n if it increases its supply as much as possible from one time step to another. Clearly, $C_a^{\max,+}[n]$ is bounded by S_a^{\max} . If $S_a^x[f(n)] > S_a^x[n]$ (case $n = n_2$ in our illustration), a can provide more reserves than in case of $S_a^x[f(n)] < S_a^x[n]$ (case $n = n_1$ in our illustration) because it can also include reserves that result from not decreasing the supply from one time step to another. . . .	193
12.1	The power plants use TBST-based schedules as a blueprint for how much power to provide in which situation. For each node, the sum of the scheduled output (upper half of the nodes) of the base load and the peaking power plant matches the corresponding expected residual load (lower half of the nodes). For instance, the base load power plant should provide 60 MW and the peaking power plant 20 MW in case of a residual load of 80 MW in t_1 (here, $t_i = t_{\text{now}} + i \cdot \Delta\tau$ with $i = \{1, 2, 3\}$). While the base load power plant should provide an output that is more or less constant and independent of changes in the residual load, the peaking power plant – whose maximum output is 20 MW – should make adjustments in accordance with residual load changes. The dotted line symbolizes the development of the selection of the most suitable scenario, i.e., the scenario the power plants identified as closest to the actual residual load, which is indicated by the light curve. Between t_{now} and t_1 , the power plants reactively switch from one scenario to another after they noticed that the demand tends to be closer to 60 MW instead of 80 MW.	202
12.2	An intermediary's basic procedure applied to compensate for deviations by means of reactive supply adjustments. The actual compensation for the deviation present in the intermediary's subsystem is performed in the activity "Coordinate Compensation for Deviations". Each time the intermediary calls this activity, a new <i>compensation round</i> is started.	204
12.3	An agent's basic procedure when participating in a compensation for deviations. The decision about the future supply is based on the demand target (if available), the deviation that has to be compensated for, the scheduled supply for different demand scenarios, and the agent's control model. Except for the first iteration of the first compensation round in t_{now} , all adjustments are based on the remaining deviation. In the second phase of the iterative compensation for deviations, an agent's sensitivity escalates from one iteration to another. In the last escalation stage, adjustments are not necessarily schedule-compliant and only restricted by the agent's control model.	208
12.4	The scheduled output for the anticipated scenario as well as the actual output of a hydro power plant over a time frame of 60 min. Recorded in our simulation environment for autonomous power systems. Despite necessary reactive supply adjustments, the actual output resembles the scheduled output.	213

List of Tables

1.1	The number, peak output, and annual feed-in of power plants subsidized by the German Renewable Energy Act (EEG) in 2010 and 2015 according to EnergyMap.info: From 2010 to 2015, these numbers increased by 113 %, 103 %, and 80 %, respectively. As for non-dispatchable power plants, we even observe an increase of 115 %, 112 %, and 96 %, respectively. But also the share of dispatchable power plants grew significantly by 39 %, 48 %, and 51 %, respectively. In the same time frame, the consumption of electricity maintained more or less constant (604950 GWh/year in 2010, compared to 608051 GWh/year in 2015). Hence, the fraction of the consumption satisfied by EEG-subsidized power plants raised from 15 % to 26 %, which is an increase of 79 %.	9
5.1	Deviations between the actual and the expected residual load obtained with different methods for determining the expectation: The expected residual load is either (1) equivalent to the predictor’s unmodified residual load prediction, (2) based on a trust value, or (3) derived from TBSs. Here, \min_δ denotes the minimum deviation between actual and expected residual load, \max_δ the maximum, μ_δ the average, and σ_δ the corresponding standard deviation.	43
5.2	Relative reduction of the deviation between the expected and the actual residual load, compared to the residual load prediction for $N = 4$ time slots. The data is based on expectations and predictions the five AVPPs provided at daytime, which comprises 60 of the 96 time steps of the fifth simulated day (we deliberately omit the results obtained at night because these data do not allow us to derive reasonable values for the relative reduction for technical reasons). All listed values are based on 300 data points. For each AVPP, we calculated the average relative reduction of the prediction error over the 60 time steps at daytime, as well as the corresponding standard deviation. Here, $\mu_\%$ denotes the mean of the five average values (one per AVPP), and $\sigma_\%$ the average of the five corresponding standard deviations. Values in parentheses denote standard deviation.	45
6.1	Evaluation results for the objectives homogeneous partitioning (HPm) and clustering (C) for different system sizes. All values are averages over 1000 experiments consisting of 300 time steps each; values in parentheses denote standard deviations. A reorganization was triggered as soon as the normalized fitness was below 0.9.	66
7.1	Selected results for the objectives C, AC, HPm, and HPs obtained with PSOPP and RDM using a time limit of 10s for different values of the number of elements. All values are averages over 500 runs. Parentheses contain standard deviations.	93

7.2	The avg. \bar{r}_n and the max. \hat{r}_n time PSOPP needed to find optimal solutions in its $500 \cdot 99\% = 495$ best runs, and the avg. raw value \bar{v} x-means obtained in its $100 \cdot 99\% = 99$ best runs limited by the time \hat{r}_n PSOPP needed to find the optimum for C ($s_{min} = n_{min} = 1, s_{max} = n_{max} = n$). Parentheses contain standard deviations.	96
8.1	The 11 different problem structures considered for SPADA's evaluation. Each problem structure defines the number n of agents participating in the partitioning problem, the parametrization of the partitioning constraints, as well as the dimensions of the optimization problem (i.e., the number of objectives).	122
8.2	Overview of the worst and the best raw values used to calculate normalized fitness values for the different objectives $o \in \{C, AC, HPm, HPs\}$ and for different numbers n of agents participating in the partitioning problem.	123
8.3	Unrestricted partitionings: Mean fitness values obtained by SPADA and PSOPP for different combinations of objectives and problem structures after a runtime of 10 s. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.	125
8.4	Restricted size of partitions: Mean fitness values obtained by SPADA and PSOPP for objective C, 1000 agents, and different restrictions of the minimum and the maximum partition size after a runtime of 10 s. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.	126
8.5	Restricted size and number of partitions: Mean fitness values obtained by SPADA and PSOPP for objective C, 1000 agents, and different restrictions of the number-of-partitions constraints after a runtime of 15 s. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.	129
8.6	First multi-objective evaluation scenario: Overview of the mean fitness values attained by SPADA and PSOPP for the six dimensions of a multi-objective setting after a runtime of 120 s. The set of objectives comprises one objective for C, one objective for HPm, and four objectives for HPs. It is performed on the basis of problem structure PS-10. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.	131
8.7	Second multi-objective evaluation scenario: Overview of the mean fitness values attained by SPADA and PSOPP for the six dimensions of a multi-objective setting after a runtime of 60 s. The set of objectives comprises one objective for AC, one objective for HPm, and four objectives for HPs. It is performed on the basis of problem structure PS-10. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.	132
8.8	Third multi-objective evaluation scenario: Overview of the mean fitness values attained by SPADA and PSOPP for the six dimensions of a multi-objective setting after a runtime of 400 s. The set of objectives comprises one objective for C, one objective for HPm, and four objectives for HPs. It is performed on the basis of problem structure PS-11. All fitness values are averaged over 100 runs. Values in parentheses denote standard deviations.	133
9.1	Evaluation results for different combinations of the maximal scheduling times and the size of partitions ("Large Partitions": $s_{min} = 8$ and $s_{max} = 80$; "Small Partitions": $s_{min} = 2$ and $s_{max} = 15$) using MaxSPAN constraints as dissolution and introduction constraints. Agents aimed for homogeneous partitionings. All data is based on 100 runs, each comprising 96 time steps. Values in parentheses denote standard deviations. To avoid that an AVPP can decrease its costs by not covering its assigned residual load, we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$ for violated assigned residual loads.	143

9.2	Evaluation results for different combinations of the maximal scheduling times and the size of partitions (“Large Partitions”: $s_{min} = 8$ and $s_{max} = 80$; “Small Partitions”: $s_{min} = 2$ and $s_{max} = 15$) using MaxSPAN constraints as dissolution and introduction constraints. In contrast to Table 9.1, agents aimed for partitionings that <i>minimize</i> the fitness of homogeneous partitioning. All data is based on 100 runs, each comprising 96 time steps. Values in parentheses denote standard deviations. To avoid that an AVPP can decrease its costs by not covering its assigned residual load, we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$ for violated assigned residual loads.	144
10.1	Evaluation results obtained by TruCAOS and RegioC for different hierarchies of AVPPs. All results are averaged over the 200 runs per evaluation scenario. Values in parentheses denote standard deviations.	173
11.1	Evaluation results for me-TBST, ma-TBST, me-1-TBS, and ma-1-TBS in combination with different hierarchical structures of AVPPs. All results are averaged over 100 simulation runs for each evaluation scenario. Values in parentheses denote standard deviations. “Prediction error” stands for the difference between the expected residual load and the actual residual load. In case of me-1-TBS and ma-1-TBS, the results for the reduction of the prediction error correspond to the reduction by the anticipated TBS of the modes me-TBST and ma-TBST, respectively.	189
11.2	Evaluation results obtained by TruCAOS and RegioC for modes E-nu (no uncertainties, optimized provision of reserves disabled), E-nr (uncertain residual load, optimized provision of reserves disabled), and E-r (uncertain residual load, optimized provision of reserves enabled) in combination with different hierarchies. “#Runs Exceeding the Max. Scheduling Time” denotes the number of RegioC runs that were aborted due to an exceeded maximal scheduling time of 10 min. All other results are averaged over the 100 runs per evaluation scenario. Values in parentheses denote standard deviations. Apart from mode E-nu, all scheduling times are presented in seconds. To avoid that an AVPP can decrease its costs by not covering its assigned residual load, we imposed a penalty of $17.50 \frac{\text{euro cent}}{\text{kWh}}$ for violated assigned residual loads.	197
12.1	Evaluation results for the four evaluation scenarios <i>ScenarioBased</i> , <i>AllEscStages</i> , <i>OnlyLastEscStage</i> , and <i>BadPrefScenario</i> . All results are averaged over the 200 simulation runs for each evaluation scenario. Values in parentheses denote standard deviations. The results for the evaluation scenario <i>1-TBS</i> are similar to those of <i>OnlyLastEscStage</i> . Differences are highlighted in the discussion.	212

Bibliography

- [1] S. Abdallah and V. R. Lesser, “Organization-Based Cooperative Coalition Formation,” in *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004)*. IEEE Computer Society, 2004, pp. 162–168.
- [2] M. Abouelela and M. El-Darieby, “Multidomain Hierarchical Resource Allocation for Grid Applications,” *Journal of Electrical and Computer Engineering*, vol. 2012, pp. 1–8, 2012.
- [3] S. Alam, G. Dobbie, and P. Riddle, “An Evolutionary Particle Swarm Optimization Algorithm for Data Clustering,” in *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2008)*. IEEE, 2008, pp. 1–6.
- [4] R. Alford, P. Bercher, and D. W. Aha, “Tight Bounds for HTN Planning,” in *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, R. I. Brafman, C. Domshlak, P. Haslum, and S. Zilberstein, Eds. AAAI Press, 2015, pp. 7–15.
- [5] M. Alrifai and T. Risse, “Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition,” in *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, Eds. ACM, 2009, pp. 881–890.
- [6] G. Anders, F. Siefert, J.-P. Steghöfer, H. Seebach, F. Nafz, and W. Reif, “Structuring and Controlling Distributed Power Sources by Autonomous Virtual Power Plants,” in *Proceedings of the Power & Energy Student Summit 2010 (PESS 2010)*, Oct. 2010, pp. 40–42.
- [7] G. Anders, H. Seebach, F. Nafz, J.-P. Steghöfer, and W. Reif, “Decentralized Reconfiguration for Self-Organizing Resource-Flow Systems Based on Local Knowledge,” in *Proceedings of the 8th IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe 2011)*, Apr. 2011, pp. 20–31.
- [8] G. Anders, C. Hinrichs, F. Siefert, P. Behrmann, W. Reif, and M. Sonnenschein, “On the Influence of Inter-Agent Variation on Multi-Agent Algorithms Solving a Dynamic Task Allocation Problem under Uncertainty,” in *Proceedings of the Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)*. IEEE Computer Society, 2012, pp. 29–38.
- [9] G. Anders, F. Siefert, J.-P. Steghöfer, and W. Reif, “A Decentralized Multi-agent Algorithm for the Set Partitioning Problem,” in *Proceedings of the 15th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2012)*, ser. Lecture Notes in Computer Science, I. Rahwan, W. Wobcke, S. Sen, and T. Sugawara, Eds., vol. 7455. Springer, 2012, pp. 107–121.
- [10] G. Anders, J.-P. Steghöfer, F. Siefert, and W. Reif, “A Trust- and Cooperation-Based Solution of a Dynamic Resource Allocation Problem,” in *Proceedings of the 7th IEEE International Conference*

- on Self-Adaptive and Self-Organizing Systems (SASO 2013)*. IEEE Computer Society, 2013, pp. 1–10.
- [11] G. Anders, A. Schiendorfer, J.-P. Steghöfer, and W. Reif, “Robust Scheduling in a Self-Organizing Hierarchy of Autonomous Virtual Power Plants,” in *27th International Conference on Architecture of Computing Systems, Workshop Proceedings (ARCS 2014)*, W. Stechele and T. Wild, Eds. VDE Verlag / IEEE Xplore, 2014, pp. 1–8.
- [12] G. Anders, F. Siefert, M. Mair, and W. Reif, “Proactive Guidance for Dynamic and Cooperative Resource Allocation under Uncertainties,” in *Proceedings of the Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2014)*. IEEE Computer Society, 2014, pp. 21–30.
- [13] G. Anders, F. Siefert, J.-P. Steghöfer, and W. Reif, “Trust-Based Scenarios – Predicting Future Agent Behavior in Open Self-organizing Systems,” in *Self-Organizing Systems – 7th IFIP TC 6 International Workshop (IWSOS 2013)*, ser. Lecture Notes in Computer Science, W. Elmenreich, F. Dressler, and V. Loreto, Eds., vol. 8221. Springer, 2014, pp. 90–102.
- [14] G. Anders, A. Schiendorfer, F. Siefert, J.-P. Steghöfer, and W. Reif, “Cooperative Resource Allocation in Open Systems of Systems,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 2, pp. 11:1–11:44, 2015.
- [15] G. Anders, F. Siefert, and W. Reif, “A Particle Swarm Optimizer for Solving the Set Partitioning Problem in the Presence of Partitioning Constraints,” in *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2015)*, S. Loiseau, J. Filipe, B. Duval, and H. J. van den Herik, Eds., vol. 2. SciTePress, 2015, pp. 151–163.
- [16] —, “A Heuristic for Constrained Set Partitioning in the Light of Heterogeneous Objectives,” in *Agents and Artificial Intelligence: 7th International Conference, ICAART 2015, Lisbon, Portugal, January 10-12, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, B. Duval, H. J. van den Herik, S. Loiseau, and J. Filipe, Eds. Springer, 2015, vol. 9494, pp. 223–244.
- [17] G. Anders, F. Siefert, A. Schiendorfer, H. Seebach, J.-P. Steghöfer, B. Eberhardinger, O. Kosak, and W. Reif, “Specification and Design of Trust-Based Open Self-Organising Systems,” in *Trustworthy Open Self-Organising Systems*, ser. Autonomic Systems, W. Reif, G. Anders, H. Seebach, J.-P. Steghöfer, E. André, J. Hähner, C. Müller-Schloer, and T. Ungerer, Eds. Springer International Publishing, 2016, pp. 17–54.
- [18] H. Appelrath, H. Kagermann, and C. Mayer, Eds., *Future Energy Grid—Migration to the Internet of Energy*, ser. acatech STUDY. Munich: acatech – National Academy of Science and Engineering, 2012.
- [19] K. R. Apt and A. Witzel, “A Generic Approach to Coalition Formation,” *International Game Theory Review (IGTR)*, vol. 11, no. 3, pp. 347–367, 2009.
- [20] A. M. Aref and T. T. Tran, “A decentralized trustworthiness estimation model for open, multiagent systems (DTMAS),” *Journal of Trust Management*, vol. 2, no. 1, pp. 1–20, 2015.
- [21] J. M. Arroyo and A. J. Conejo, “Modeling of Start-Up and Shut-Down Power Trajectories of Thermal Units,” *IEEE Transactions on Power Systems*, vol. 19, no. 3, pp. 1562–1568, Aug. 2004.
- [22] A. Artikis and J. V. Pitt, “Specifying Open Agent Systems: A Survey,” in *Proceedings of the 9th International Workshop on Engineering Societies in the Agents World IX (ESAW 2008), Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Artikis, G. Picard, and L. Vercouter, Eds., vol. 5485. Springer, 2008, pp. 29–45.
- [23] W. R. Ashby, *Facets of Systems Science*. Boston, MA: Springer US, 1991, ch. Principles of the Self-Organizing System, pp. 521–536.

-
- [24] S. Äyrämö and T. Kärkkäinen, "Introduction to partitioning-based clustering methods with a robust example," Reports of the Department of Mathematical Information Technology, Series C. Software and Computational Engineering of the University of Jyväskylä, Tech. Rep., 2006.
 - [25] S. Ba and P. A. Pavlou, "Evidence of the Effect of Trust Building Technology in Electronic Markets: Price Premiums and Buyer Behavior," *MIS Quarterly*, vol. 26, no. 3, pp. 243–268, 2002.
 - [26] S. Ba, A. B. Whinston, and H. Zhang, "Building Trust in the Electronic Market Through an Economic Incentive Mechanism," in *Proceedings of the Twentieth International Conference on Information Systems (ICIS 1999)*, P. De and J. I. DeGross, Eds. Association for Information Systems, 1999, pp. 208–213.
 - [27] E. Balas and M. W. Padberg, "Set Partitioning: A Survey," *SIAM Review*, vol. 18, no. 4, pp. 710–760, 1976.
 - [28] S. Bandyopadhyay and E. J. Coyle, "An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*. IEEE, 2003.
 - [29] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, "A Unified Approach to Approximating Resource Allocation and Scheduling," *Journal of the ACM (JACM)*, vol. 48, no. 5, pp. 1069–1090, 2001.
 - [30] G. Beavers and H. Hexmoor, "Teams of Agents," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, 2001, pp. 574–582.
 - [31] I. Bel, A. Valenti, J. Maire, J. M. Corera, and P. Lang, "Innovative Operation with Aggregated Distributed Generation," in *Proceedings of the 19th International Conference on Electricity Distribution*, 2007.
 - [32] A. Ben-Tal and A. Nemirovski, "Selected Topics in Robust Convex Optimization," *Mathematical Programming*, vol. 112, no. 1, pp. 125–158, 2007.
 - [33] C. Bender, D. Brody, and B. Meister, "Quantum field theory of partitions," *Journal of Mathematical Physics*, vol. 40, no. 7, p. 3239, 1999.
 - [34] A. Benonysson, B. Bøhm, and H. F. Ravn, "Operational optimization in a district heating system," *Energy Conversion and Management*, vol. 36, no. 5, pp. 297–314, 1995.
 - [35] R. Bent, C. Coffrin, R. Gumucio, and P. Van Hentenryck, "Transmission Network Expansion Planning: Bridging the Gap between AC Heuristics and DC Approximations," in *Proceedings of the Power Systems Computation Conference (PSCC)*, Aug. 2014, pp. 1–8.
 - [36] P. Berkhin, "A Survey of Clustering Data Mining Techniques," in *Grouping Multidimensional Data – Recent Advances in Clustering*, J. Kogan, C. K. Nicholas, and M. Teboulle, Eds. Springer, 2006, pp. 25–71.
 - [37] Y. Bernard, L. Klejnowski, J. Hähner, and C. Müller-Schloer, "Towards Trust in Desktop Grid Systems," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010)*. IEEE Computer Society, 2010, pp. 637–642.
 - [38] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. Springer-Verlag New York, Inc., 2006, vol. 1.
 - [39] S. Boon and J. Holmes, "The dynamics of interpersonal trust: Resolving uncertainty in the face of risk," in *Cooperation and prosocial behaviour*, R. Hinde and J. Groebel, Eds. Cambridge, UK: Cambridge University Press, 1991, pp. 190–211.

- [40] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou, "Hierarchical Scheduling Framework Based on Compositional Analysis Using Uppaal," in *Formal Aspects of Component Software – 10th International Symposium (FACS 2013)*, ser. Lecture Notes in Computer Science, J. L. Fiadeiro, Z. Liu, and J. Xue, Eds., vol. 8348. Springer, 2013, pp. 61–78.
- [41] F. Bouffard and F. D. Galiana, "Stochastic Security for Operations Planning With Significant Wind Power Generation," *IEEE Transactions on Power Systems*, vol. 23, no. 2, pp. 306–316, May 2008.
- [42] J. Branke, *Evolutionary Optimization in Dynamic Environments*, ser. Genetic Algorithms and Evolutionary Computation. Norwell, MA, USA: Kluwer Academic Publishers, 2001, vol. 3.
- [43] S. Breban and J. Vassileva, "A Coalition Formation Mechanism Based on Inter-Agent Trust Relationships," in *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*. ACM, 2002, pp. 306–307.
- [44] S. A. Brueckner, T. C. Belding, R. Bisson, E. Downs, and H. V. D. Parunak, "Swarming Polyagents Executing Hierarchical Task Networks," in *Proceedings of the 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009)*. IEEE Computer Society, 2009, pp. 51–60.
- [45] F. Buccafurri, D. Rosaci, G. M. L. Sarnè, and D. Ursino, "An Agent-Based Hierarchical Clustering Approach for E-commerce Environments," in *Proceedings of the Third International Conference on E-Commerce and Web Technologies (EC-Web 2002)*, ser. Lecture Notes in Computer Science, K. Bauknecht, A. M. Tjoa, and G. Quirchmayr, Eds., vol. 2455. Springer, 2002, pp. 109–118.
- [46] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Network Robustness and Fragility: Percolation on Random Graphs," *Physical Review Letters*, vol. 85, pp. 5468–5471, Dec. 2000.
- [47] A. Campbell, C. Riggs, and A. S. Wu, "On the Impact of Variation on Self-Organizing Systems," in *Proceedings of the 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2011)*. IEEE Computer Society, 2011, pp. 119–128.
- [48] G. Chalkiadakis, V. Robu, R. Kota, A. Rogers, and N. R. Jennings, "Cooperatives of Distributed Energy Resources for Efficient Virtual Power Plants," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '11)*, L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, Eds., vol. 2, 2011, pp. 787–794.
- [49] H. Chan and A. Perrig, "ACE: An Emergent Algorithm for Highly Uniform Cluster Formation," in *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN 2004)*, ser. Lecture Notes in Computer Science, H. Karl, A. Willig, and A. Wolisz, Eds., vol. 2920. Springer, 2004, pp. 154–171.
- [50] B. Chang, S. Kuo, Y. Liang, and D. Wang, "Markov Chain-Based Trust Model for Analyzing Trust Value in Distributed Multicasting Mobile Ad Hoc Networks," in *Proceedings of the 3rd IEEE Asia-Pacific Services Computing Conference (APSCC 2008)*. IEEE Computer Society, 2008, pp. 156–161.
- [51] Y. Chevalere, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. A. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa, "Issues in Multiagent Resource Allocation," *Informatica (Slovenia)*, vol. 30, no. 1, pp. 3–31, 2006.
- [52] P. C. Chu and J. E. Beasley, "Constraint Handling in Genetic Algorithms: The Set Partitioning Problem," *Journal of Heuristics*, vol. 4, no. 4, pp. 323–357, 1998.

- [53] A. W. Colombo, R. Schoop, and R. Neubert, "An Agent-Based Intelligent Control Platform for Industrial Holonic Manufacturing Systems," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 1, pp. 322–337, 2005.
- [54] S. Consoli, J. A. Moreno-Pérez, K. Darby-Dowman, and N. Mladenovic, "Discrete Particle Swarm Optimization for the minimum labelling Steiner tree problem," *Natural Computing*, vol. 9, no. 1, pp. 29–46, 2010.
- [55] C. L. Corritore, B. Kracher, and S. Wiedenbeck, "On-line trust: concepts, evolving themes, a model," *International Journal of Human-Computer Studies*, vol. 58, no. 6, pp. 737–758, 2003.
- [56] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, and A. Koukam, "ASPECS: an Agent-oriented Software Process for Engineering Complex Systems," *Autonomous Agents and Multi-Agent Systems*, vol. 20, no. 2, pp. 260–304, 2010.
- [57] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning," *Foundations and Trends in Computer Graphics and Vision*, vol. 7, no. 2-3, pp. 81–227, 2012.
- [58] R. K. Dash, N. R. Jennings, and D. C. Parkes, "Computational-Mechanism Design: A Call to Arms," *IEEE Intelligent Systems*, vol. 18, no. 6, pp. 40–47, 2003.
- [59] R. K. Dash, S. D. Ramchurn, and N. R. Jennings, "Trust-Based Mechanism Design," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*. IEEE Computer Society, 2004, pp. 748–755.
- [60] R. K. Dash, P. Vytelingum, A. Rogers, E. David, and N. R. Jennings, "Market-Based Task Allocation Mechanisms for Limited-Capacity Suppliers," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 37, no. 3, pp. 391–405, 2007.
- [61] M. Densing, "Hydro-Electric Power Plant Dispatch-Planning—Multi-Stage Stochastic Programming with Time-Consistent Constraints on Risk," Ph.D. dissertation, ETH Zurich, Switzerland, 2007.
- [62] F. Dötsch, J. Denzinger, H. Kasinger, and B. Bauer, "Decentralized Real-time Control of Water Distribution Networks Using Self-organizing Multi-Agent Systems," in *Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*. IEEE Computer Society, 2010, pp. 223–232.
- [63] B. Eberhardinger, H. Seebach, A. Knapp, and W. Reif, "Towards Testing Self-organizing, Adaptive Systems," in *Testing Software and Systems – 26th IFIP WG 6.1 International Conference (ICTSS 2014)*, ser. Lecture Notes in Computer Science, M. G. Merayo and E. M. de Oca, Eds., vol. 8763. Springer, 2014, pp. 180–185.
- [64] B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, and W. Reif, "A Research Overview and Evaluation of Performance Metrics for Self-Organization Algorithms," in *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO Workshops (SASOW 2015)*. IEEE Computer Society, 2015, pp. 122–127.
- [65] B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, A. Knapp, and W. Reif, "An Approach for Isolated Testing of Self-Organization Algorithms," in *Software Engineering for Self-Adaptive Systems 3*, ser. Lecture Notes in Computer Science, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Springer, 2017, vol. 9640.
- [66] R. Falcone and C. Castelfranchi, "Social Trust: A Cognitive Approach," in *Trust and Deception in Virtual Societies*, C. Castelfranchi and Y.-H. Tan, Eds. Dordrecht: Springer Netherlands, 2001, pp. 55–90.

- [67] M. A. A. Faruque, R. Krist, and J. Henkel, “ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication,” in *Proceedings of the 45th Design Automation Conference (DAC 2008)*, L. Fix, Ed. ACM, 2008, pp. 760–765.
- [68] K. Fielding and D. Ruck, “Spatio-Temporal Pattern Recognition Using Hidden Markov Models,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 4, pp. 1292–1300, Oct. 1995.
- [69] S. Frey, A. Diaconescu, D. Menga, and I. M. Demeure, “A Generic Holonic Control Architecture for Heterogeneous Multiscale and Multiobjective Smart Microgrids,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 2, pp. 9:1–9:21, 2015.
- [70] F. J. M. García and J. A. M. Pérez, “Jumping Frogs Optimization: a new swarm method for discrete optimization,” Documentos de Trabajo del DEIOC, Department of Statistics, O.R. and Computing, University of La Laguna, Tenerife, Spain, Tech. Rep. 3, 2008.
- [71] J. García-González, R. M. R. de la Muela, L. M. Santos, and A. M. González, “Stochastic Joint Optimization of Wind Generation and Pumped-Storage Units in an Electricity Market,” *IEEE Transactions on Power Systems*, vol. 23, no. 2, pp. 460–468, May 2008.
- [72] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [73] I. P. Gent, K. E. Petrie, and J. Puget, “Symmetry in Constraint Programming,” in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, vol. 2, pp. 329–376.
- [74] D. Good, “Individuals, Interpersonal Relations, and Trust,” in *Trust: Making and Breaking Cooperative Relations*, D. Gambetta, Ed. Department of Sociology, University of Oxford, 2000, pp. 31–48.
- [75] B. Gopalakrishnan and E. L. Johnson, “Airline Crew Scheduling: State-of-the-Art,” *Annals of Operations Research*, vol. 140, no. 1, pp. 305–337, 2005.
- [76] W. A. Greene, “Partitioning Sets with Genetic Algorithms,” in *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*, J. N. Etheredge and B. Z. Manaris, Eds. AAAI Press, 2000, pp. 102–106.
- [77] N. Griffiths and M. Luck, “Coalition Formation through Motivation and Trust,” in *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*. ACM, 2003, pp. 17–24.
- [78] N. Gröwe-Kuska, H. Heitsch, and W. Römis, “Scenario Reduction and Scenario Tree Construction for Power Management Problems,” in *IEEE Power Tech Conference Proceedings*, vol. 3, June 2003.
- [79] S. Guha, R. Rastogi, and K. Shim, “Cure: An Efficient Clustering Algorithm for Large Databases,” *Information Systems*, vol. 26, no. 1, pp. 35–58, 2001.
- [80] M. R. Hassan and B. Nath, “Stock Market Forecasting Using Hidden Markov Model: A New Approach,” in *Proceedings of the Fifth International Conference on Intelligent Systems Design and Applications (ISDA 2005)*. IEEE Computer Society, 2005, pp. 192–196.
- [81] M. Hayn, V. Bertsch, and W. Fichtner, “Electricity load profiles in Europe: The importance of household segmentation,” *Energy Research & Social Science*, vol. 3, pp. 30 – 45, 2014.
- [82] R. He, J. Niu, and G. Zhang, “CBTM: A Trust Model with Uncertainty Quantification and Reasoning for Pervasive Computing,” in *Proceedings of the Third International Symposium on Parallel and Distributed Processing and Applications (ISPA 2005)*, ser. Lecture Notes in Computer Science, Y. Pan, D. Chen, M. Guo, J. Cao, and J. Dongarra, Eds., vol. 3758. Springer, 2005, pp. 541–552.

-
- [83] P. V. Hentenryck and R. Bent, *Online Stochastic Combinatorial Optimization*. The MIT Press, 2009.
 - [84] J. S. Heo, K. Y. Lee, and R. Garduno-Ramirez, "Multiobjective Control of Power Plants Using Particle Swarm Optimization Techniques," *IEEE Transactions on Energy Conversion*, vol. 21, no. 2, pp. 552–561, June 2006.
 - [85] R. Hermoso, H. Billhardt, and S. Ossowski, *Coordination, Organizations, Institutions, and Norms in Agent Systems II: AAMAS 2006 and ECAI 2006 International Workshops, COIN 2006, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. Integrating Trust in Virtual Organisations, pp. 19–31.
 - [86] L. Hernández, C. Baladrón, J. M. Aguiar, B. Carro, A. Sánchez-Esguevillas, J. Lloret, D. Chinarro, J. J. Gomez-Sanz, and D. Cook, "A Multi-Agent System Architecture for Smart Grid Management and Forecasting of Energy Demand in Virtual Power Plants," *IEEE Communications Magazine*, vol. 51, no. 1, pp. 106–113, 2013.
 - [87] C. Hewitt, "Open Information Systems Semantics for Distributed Artificial Intelligence," *Artificial Intelligence*, vol. 47, no. 1-3, pp. 79–106, 1991.
 - [88] F. Heylighen, "The Science of Self-Organization and Adaptivity," in *Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS*. Publishers Co. Ltd, 1999, pp. 253–280.
 - [89] C. Hinrichs, M. Sonnenschein, and S. Lehnhoff, "Evaluation of a Self-organizing Heuristic for Interdependent Distributed Search Spaces," in *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*, J. Filipe and A. L. N. Fred, Eds., vol. 1. SciTePress, 2013, pp. 25–34.
 - [90] R. Hochreiter and G. C. Pflug, "Financial scenario generation for stochastic multi-stage decision processes as facility location problems," *Annals of Operations Research*, vol. 152, no. 1, pp. 257–272, 2007.
 - [91] K. L. Hoffman and M. Padberg, "Solving Airline Crew Scheduling Problems by Branch-and-Cut," *Management Science*, vol. 39, no. 6, pp. 657–682, June 1993.
 - [92] B. Horling and V. R. Lesser, "A Survey of Multi-Agent Organizational Paradigms," *The Knowledge Engineering Review*, vol. 19, no. 4, pp. 281–316, 2004.
 - [93] K. Høyland and S. W. Wallace, "Generating Scenario Trees for Multistage Decision Problems," *Management Science*, vol. 47, no. 2, pp. 295–307, 2001.
 - [94] M. Hundt, R. Barth, N. Sun, S. Wissel, and A. Voß, "Verträglichkeit von erneuerbaren Energien und Kernenergie im Erzeugungsportfolio," *Technisch-ökonomische Aspekte. Studie des Instituts für Energiewirtschaft und rationelle Energieanwendung (IER) im Auftrag der E. ON Energie AG. Stuttgart*, 2009, in German.
 - [95] F. K. Hussain, E. Chang, and T. S. Dillon, "Markov Model for Modelling and Managing Dynamic Trust," in *Proceedings of the 3rd IEEE International Conference on Industrial Informatics (INDIN '05)*, Aug. 2005, pp. 725–733.
 - [96] F. K. Hussain, E. Chang, and O. K. Hussain, "A Robust Methodology for Prediction of Trust and Reputation Values," in *Proceedings of the 5th ACM Workshop On Secure Web Services (SWS 2008)*, E. Damiani and S. Proctor, Eds. ACM, 2008, pp. 97–108.
 - [97] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, "An integrated trust and reputation model for open multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 119–154, 2006.

- [98] T. Ishioka, “An Expansion of X-Means for Automatically Determining the Optimal Number of Clusters – Progressive Iterations of K-Means and Merging of the Clusters,” in *Proceedings of the International Conference on Computational Intelligence (IASTED)*, M. H. Hamza, Ed. IASTED/ACTA Press, 2005, pp. 91–96.
- [99] K. Jones, “Trust as an Affective Attitude,” *Ethics*, vol. 107, no. 1, pp. pp. 4–25, 1996.
- [100] C. M. Jonker and J. Treur, “Formal Analysis of Models for the Dynamics of Trust Based on Experiences,” in *Multi-Agent System Engineering, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW ’99)*, ser. Lecture Notes in Computer Science, F. J. Garijo and M. Boman, Eds., vol. 1647. Springer, 1999, pp. 221–231.
- [101] A. Jøsang, “A Logic for Uncertain Probabilities,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 9, no. 3, pp. 279–212, 2001.
- [102] A. Jøsang and S. Pope, “Semantic Constraints for Trust Transitivity,” in *Conceptual Modelling, Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*, ser. CRPIT, S. Hartmann and M. Stumptner, Eds., vol. 43. Australian Computer Society, 2005, pp. 59–68.
- [103] E. Kaddoum, C. Raibulet, J. Georgé, G. Picard, and M. P. Gleizes, “Criteria for the Evaluation of Self-* Systems,” in *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010)*, R. de Lemos and M. Pezzè, Eds. ACM, 2010, pp. 29–38.
- [104] A. Kamper and A. Eßer, “Strategies for Decentralised Balancing Power,” in *Biologically-inspired Optimisation Methods – Parallel Algorithms, Systems and Applications*, ser. Studies in Computational Intelligence, A. Lewis, S. Mostaghim, and M. Randall, Eds. Springer, 2009, vol. 210, pp. 261–289.
- [105] A. Kamper and H. Schmeck, “Adaptives verteiltes Lastmanagement in Bilanzkreisen,” *Informatik Spektrum*, vol. 35, no. 2, pp. 102–111, 2012, in German.
- [106] L. Kaufman and P. Rousseeuw, *Clustering by Means of Medoids*. North-Holland, 1987.
- [107] J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, Nov. 1995, pp. 1942–1948.
- [108] J. Kennedy and R. C. Eberhart, “A Discrete Binary Version of the Particle Swarm Algorithm,” in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation.*, vol. 5, Oct. 1997, pp. 4104–4108.
- [109] J. Kennedy and R. Mendes, “Neighborhood Topologies in Fully-Informed and Best-Of-Neighborhood Particle Swarms,” in *Proceedings of the 2003 IEEE International Workshop on Soft Computing in Industrial Applications (SMCia/03)*, June 2003, pp. 45–50.
- [110] J. O. Kephart and D. M. Chess, “The Vision of Autonomic Computing,” *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
- [111] R. Kiefhaber, S. Hammer, B. Savs, J. Schmitt, M. Roth, F. Kluge, E. André, and T. Ungerer, “The Neighbor-Trust Metric to Measure Reputation in Organic Computing Systems,” in *Proceedings of the Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems, Workshops Proceedings (SASOW 2011)*. IEEE Computer Society, 2011, pp. 41–46.
- [112] R. Kiefhaber, G. Anders, F. Siefert, T. Ungerer, and W. Reif, “Confidence as a Means to Assess the Accuracy of Trust Values,” in *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012)*, G. Min, Y. Wu, L. C. Liu, X. Jin, S. A. Jarvis, and A. Y. Al-Dubai, Eds. IEEE Computer Society, 2012, pp. 690–697.

- [113] R. Kiefhaber, R. Jahr, N. Msadek, and T. Ungerer, "Ranking of Direct Trust, Confidence, and Reputation in an Abstract System with Unreliable Components," in *Proceedings of the 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing (UIC/ATC 2013)*. IEEE, 2013, pp. 388–395.
- [114] M. Kim and K. S. Candan, "SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices," *Data & Knowledge Engineering*, vol. 72, pp. 285–303, 2012.
- [115] J. S. Kinnebrew and G. Biswas, "Efficient Allocation of Hierarchically-Decomposable Tasks in a Sensor Web Contract Net," in *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2009)*. IEEE Computer Society, 2009, pp. 225–232.
- [116] R. Kinney, P. Crucitti, R. Albert, and V. Latora, "Modeling cascading failures in the North American power grid," *The European Physical Journal B – Condensed Matter and Complex Systems*, vol. 46, no. 1, pp. 101–107, 2005.
- [117] L. Klejnowski, "Trusted Community: A Novel Multiagent Organisation for Open Distributed Systems," Ph.D. dissertation, Leibniz Universität Hannover, Germany, 2014.
- [118] L. Klejnowski, Y. Bernard, G. Anders, C. Müller-Schloer, and W. Reif, "Trusted Community – A Trust-based Multi-Agent Organisation for Open Systems," in *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*, J. Filipe and A. L. N. Fred, Eds., vol. 1. SciTePress, 2013, pp. 312–317.
- [119] P. Klemperer, "What Really Matters in Auction Design," *The Journal of Economic Perspectives*, vol. 16, no. 1, pp. 169–189, 2002.
- [120] A. Koestler, *The Ghost In The Machine*. Hutchinson, London, 1967.
- [121] J. K. Kok, C. J. Warmer, and I. G. Kamphuis, "PowerMatcher: Multiagent Control in the Electricity Infrastructure," in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005) – Special Track for Industrial Applications*, M. Pechoucek, D. Steiner, and S. G. Thompson, Eds. ACM, 2005, pp. 75–82.
- [122] M. Koller, "Risk as a Determinant of Trust," *Basic and Applied Social Psychology*, vol. 9, no. 4, pp. 265–276, 1988.
- [123] O. Kosak, G. Anders, F. Siefert, and W. Reif, "An Approach to Robust Resource Allocation in Large-Scale Systems of Systems," in *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2015)*. IEEE Computer Society, 2015, pp. 1–10.
- [124] C. Kost, J. N. Mayer, J. Thomsen, N. Hartmann, C. Senkpiel, S. Philipps, S. Nold, S. Lude, N. Saad, and T. Schlegl, "Levelized Cost of Electricity – Renewable Energy Technologies," *Techno-Economic Assessment of Energy Technologies, Fraunhofer ISE*, 2013.
- [125] V. Kotov, "Systems of Systems as Communicating Structures," in *Object-oriented Technology and Computing Systems Re-engineering*, H. Zedan and A. Cau, Eds. Chichester, USA: Horwood Publishing, Ltd., 1999, pp. 141–154.
- [126] Y. Kudo and T. Murai, "On a Criterion of Similarity between Partitions Based on Rough Set Theory," in *Proceedings of the 12th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing, (RSFDGrC 2009)*, ser. Lecture Notes in Computer Science, H. Sakai, M. K. Chakraborty, A. E. Hassanien, D. Slezak, and W. Zhu, Eds., vol. 5908. Springer, 2009, pp. 101–108.

- [127] U. Kuter and J. Golbeck, "Using Probabilistic Confidence Models for Trust Inference in Web-Based Social Networks," *ACM Transactions on Internet Technology*, vol. 10, no. 2, pp. 8:1–8:23, 2010.
- [128] W. H. Kwon and S. H. Han, *Receding Horizon Control: Model Predictive Control for State Models*. Springer Science & Business Media, 2006.
- [129] R. Lamarche-Perrin, Y. Demazeau, and J. Vincent, "A Generic Algorithmic Framework to Solve Special Versions of the Set Partitioning Problem," in *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014)*. IEEE Computer Society, 2014, pp. 891–897.
- [130] C. Lassner and R. Lienhart, "The fertilized forests Decision Forest Library," in *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference (MM '15)*, X. Zhou, A. F. Smeaton, Q. Tian, D. C. A. Bulterman, H. T. Shen, K. Mayer-Patel, and S. Yan, Eds. ACM, 2015, pp. 681–684.
- [131] J. Li, G. Poulton, and G. James, "Coordination of Distributed Energy Resource Agents," *Applied Artificial Intelligence*, vol. 24, no. 5, pp. 351–380, 2010.
- [132] L. Li, Y. Wang, and V. Varadharajan, "Fuzzy Regression Based Trust Prediction in Service-Oriented Applications," in *Proceedings of the 6th International Conference on Autonomic and Trusted Computing (ATC 2009)*, ser. Lecture Notes in Computer Science, W. Reif, G. Wang, and J. Indulska, Eds., vol. 5586. Springer, 2009, pp. 221–235.
- [133] X. Li, "Adaptively Choosing Neighbourhood Bests Using Species in a Particle Swarm Optimizer for Multimodal Function Optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, ser. Lecture Notes in Computer Science, K. Deb, R. Poli, W. Banzhaf, H. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, Eds., vol. 3102. Springer, 2004, pp. 105–116.
- [134] X. Lin, S. L. Janak, and C. A. Floudas, "A new robust optimization approach for scheduling under uncertainty: : I. Bounded uncertainty," *Computers & Chemical Engineering*, vol. 28, no. 6-7, pp. 1069–1085, 2004.
- [135] J. Liu and M. Zhou, "Tree-Assisted Gossiping for Overlay Video Distribution," *Multimedia Tools and Applications*, vol. 29, no. 3, pp. 211–232, 2006.
- [136] P. Lombardi, M. Powalko, and K. Rudion, "Optimal Operation of a Virtual Power Plant," in *Power Energy Society General Meeting (PES '09)*, July 2009, pp. 1–6.
- [137] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 1-4, pp. 72–93, 2005.
- [138] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.
- [139] M. S. Mahmoud, "Multilevel Systems Control and Applications: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 3, pp. 125–143, 1977.
- [140] W. H. J. Manthorpe, "The Emerging Joint System of Systems: A Systems Engineering Challenge and Opportunity for APL," *John Hopkins APL Technical Digest*, pp. 305–310, 1996.
- [141] S. P. Marsh, "Formalising Trust as a Computational Concept," Ph.D. dissertation, University of Stirling, United Kingdom, 1994.

-
- [142] R. C. Mayer, J. H. Davis, and F. D. Schoorman, “An Integrative Model of Organizational Trust,” *The Academy of Management Review*, vol. 20, no. 3, pp. pp. 709–734, 1995.
 - [143] S. D. McArthur, E. M. Davidson, V. M. Catterson, A. L. Dimeas, N. D. Hatziaargyriou, F. Ponci, and T. Funabashi, “Multi-Agent Systems for Power Engineering Applications – Part I: Concepts, Approaches, and Technical Challenges,” *IEEE Transactions on Power Systems*, vol. 22, no. 4, pp. 1743–1752, 2007.
 - [144] D. McKnight, L. Cummings, and N. Chervany, “Initial trust formation in new organizational relationships,” *The Academy of Management Review*, vol. 23, no. 3, pp. 473–490, 1998.
 - [145] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics (Second Edition)*. Springer, 2004.
 - [146] V. Miranda, H. Keko, and A. Jaramillo, *Advances in Evolutionary Computing for System Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. EPSO: Evolutionary Particle Swarms, pp. 139–167.
 - [147] T. Mora, A. B. Sesay, J. Denzinger, H. Golshan, G. Poissant, and C. Konecnik, “Cooperative search for optimizing pipeline operations,” in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008) – Industry and Applications Track*, M. Berger, B. Burg, and S. Nishiyama, Eds. IFAAMAS, 2008, pp. 115–122.
 - [148] N. Msadek, R. Kieffhaber, and T. Ungerer, “A trustworthy, fault-tolerant and scalable self-configuration algorithm for Organic Computing systems,” *Journal of Systems Architecture – Embedded Systems Design*, vol. 61, no. 10, pp. 511–519, 2015.
 - [149] —, “Trustworthy Self-optimization in Organic Computing Environments,” in *Proceedings of the 28th International Conference on Architecture of Computing Systems (ARCS 2015)*, ser. Lecture Notes in Computer Science, L. M. Pinho, W. Karl, A. Cohen, and U. Brinkschulte, Eds., vol. 9017. Springer, 2015, pp. 123–134.
 - [150] L. Mui, M. Mohtashemi, and A. Halberstadt, “A Computational Model of Trust and Reputation,” in *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35 2002)*. IEEE Computer Society, 2002, p. 188.
 - [151] C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., *Organic Computing – A Paradigm Shift for Complex Systems*. Springer, 2011.
 - [152] J. Mur-Amada and A. Bayocli-Rujula, “Wind Power Variability Model Part II – Probabilistic Power Flow,” in *Proceedings of the 9th International Conference on Electrical Power Quality and Utilisation (EPQU 2007)*, Oct. 2007, pp. 1–6.
 - [153] F. Nafz, H. Seebach, J.-P. Steghöfer, G. Anders, and W. Reif, “Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach,” in *Organic Computing – A Paradigm Shift for Complex Systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer, 2011, pp. 79–93.
 - [154] P. H. Nguyen, W. L. Kling, and P. F. Ribeiro, “Agent-Based Power Routing in Active Distribution Networks,” in *Proceedings of the 2nd IEEE PES International Conference and Exhibition on “Innovative Smart Grid Technologies” (ISGT Europe 2011)*. IEEE, 2011, pp. 1–6.
 - [155] A. Nieße and M. Sonnenschein, “Using Grid Related Cluster Schedule Resemblance for Energy Rescheduling – Goals and Concepts for Rescheduling of Clusters in Decentralized Energy Systems,” in *Proceedings of the 2nd International Conference on Smart Grids and Green IT Systems (SMART-GREENS 2013)*, B. Donnellan, J. F. Martins, M. Helfert, and K. Krempels, Eds. SciTePress, 2013, pp. 22–31.

- [156] —, “A Fully Distributed Continuous Planning Approach for Decentralized Energy Units,” in *45. Jahrestagung der Gesellschaft für Informatik, Informatik 2015*, ser. LNI, D. W. Cunningham, P. Hofstedt, K. Meer, and I. Schmitt, Eds., vol. 246. GI, 2015, pp. 151–165.
- [157] A. Nieße, S. Lehnhoff, M. Tröschel, M. Uslar, C. Wissing, H. Appelrath, and M. Sonnenschein, “Market-Based Self-Organized Provision of Active Power and Ancillary Services: An Agent-Based Approach for Smart Distribution Grids,” in *Complexity in Engineering (COMPENG 2012)*. IEEE, 2012, pp. 1–5.
- [158] Office of Electric Transmission and Distribution, ““Grid 2030” – a National Vision for Electricity’s Second 100 Years,” US Department of Energy, Tech. Rep., 2003.
- [159] E. Ogston, B. J. Overeinder, M. van Steen, and F. M. T. Brazier, “A Method for Decentralized Clustering in Large Multi-Agent Systems,” in *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*. ACM, 2003, pp. 789–796.
- [160] M. G. H. Omran, A. A. Salman, and A. P. Engelbrecht, “Dynamic clustering using particle swarm optimization with application in image segmentation,” *Pattern Analysis and Applications*, vol. 8, no. 4, pp. 332–344, 2006.
- [161] N. P. Padhy, “Unit Commitment—A Bibliographical Survey,” *IEEE Transactions on Power Systems*, vol. 19, no. 2, pp. 1196–1205, May 2004.
- [162] V. S. Pappala and I. Erlich, “Power System Optimization under Uncertainties: A PSO Approach,” in *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2008)*. IEEE, 2008, pp. 1–8.
- [163] J. Pitt and A. Nowak, “The Reinvention of Social Capital for Socio-Technical Systems [Special Section Introduction],” *IEEE Technology and Society Magazine*, vol. 33, no. 1, pp. 27–33, 2014.
- [164] J. Pitt, J. Schaumeier, D. Busquets, and S. Macbeth, “Self-Organising Common-Pool Resource Allocation and Canons of Distributive Justice,” in *Proceedings of the Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)*. IEEE Computer Society, 2012, pp. 119–128.
- [165] R. Porter, A. Ronen, Y. Shoham, and M. Tennenholtz, “Mechanism Design with Execution Uncertainty,” in *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI ’02)*, A. Darwiche and N. Friedman, Eds. Morgan Kaufmann, 2002, pp. 414–421.
- [166] E. Pournaras, M. Warnier, and F. Brazier, “Adaptive Agent-Based Self-Organization for Robust Hierarchical Topologies,” in *Proceedings of the International Conference on Adaptive and Intelligent Systems (ICAIS ’09)*, Sept. 2009, pp. 69–76.
- [167] H. Prothmann, “Organic Traffic Control,” Ph.D. dissertation, Karlsruhe Institute of Technology, Germany, 2011.
- [168] D. Pudjianto, C. Ramsay, and G. Strbac, “Virtual power plant and system integration of distributed energy resources,” *IET Renewable Power Generation*, vol. 1, no. 1, pp. 10–16, Mar. 2007.
- [169] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, “A Distributed Algorithm for Large-Scale Graph Partitioning,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 2, pp. 12:1–12:24, 2015.
- [170] T. Rahwan, S. D. Ramchurn, N. R. Jennings, and A. Giovannucci, “An Anytime Algorithm for Optimal Coalition Structure Generation,” *Journal of Artificial Intelligence Research (JAIR)*, vol. 34, pp. 521–567, 2009.

-
- [171] S. D. Ramchurn, T. D. Huynh, and N. R. Jennings, "Trust in multi-agent systems," *The Knowledge Engineering Review*, vol. 19, no. 1, pp. 1–25, 2004.
 - [172] S. D. Ramchurn, N. R. Jennings, C. Sierra, and L. Godo, "Devising A Trust Model For Multi-Agent Interactions Using Confidence And Reputation," *Applied Artificial Intelligence*, vol. 18, no. 9-10, pp. 833–852, 2004.
 - [173] S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. R. Jennings, "Putting the 'Smarts' into the Smart Grid: A Grand Challenge for Artificial Intelligence," *Communications of the ACM*, vol. 55, no. 4, pp. 86–97, 2012.
 - [174] L. Rani, M. Mam, and S. Kumar, "Economic Load Dispatch In Thermal Power Plant Taking Real Time Efficiency As An Additional Constraints," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 7, 2013.
 - [175] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck, "Towards a generic observer/controller architecture for Organic Computing," in *Informatik 2006 – Informatik für Menschen, Band 1, Beiträge der 36. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, ser. LNI, C. Hochberger and R. Liskowsky, Eds., vol. 93. GI, 2006, pp. 112–119.
 - [176] C. Rosinger and S. Beer, "Glaubwürdigkeit in dynamischen Wirkleistungsverbünden," *Informatik Spektrum*, vol. 38, no. 2, pp. 103–110, 2015.
 - [177] D. M. Rousseau, S. B. Sitkin, R. S. Burt, and C. Camerer, "Not so different after all: A cross-discipline view of trust," *Academy of Management Review*, vol. 23, no. 3, pp. 393–404, 1998.
 - [178] P. Ruiz, C. Philbrick, E. Zak, K. Cheung, and P. Sauer, "Uncertainty Management in the Unit Commitment Problem," *IEEE Transactions on Power Systems*, vol. 24, no. 2, pp. 642–651, May 2009.
 - [179] S. Ruthe, C. Rehtanz, and S. Lehnhoff, "A market-oriented stochastic optimization framework and its application in the energy domain," in *Industrial Electronics Society, IECON 2013 – 39th Annual Conference of the IEEE*, Nov. 2013, pp. 4721–4726.
 - [180] W. Saad, Z. Han, M. Debbah, A. Hjørungnes, and T. Basar, "Coalitional Game Theory for Communication Networks: A Tutorial," *IEEE Signal Processing Magazine*, vol. 26, no. 5, pp. 77–97, 2009.
 - [181] J. Sabater and C. Sierra, "Social ReGreT, a reputation model based on social relations," *SIGecom Exchanges*, vol. 3, no. 1, pp. 44–56, 2002.
 - [182] A. P. Sage and C. D. Cuppan, "On the Systems Engineering and Management of Systems of Systems and Federations of Systems," *Information, Knowledge, Systems Management*, vol. 2, no. 4, pp. 325–345, 2001.
 - [183] N. V. Sahinidis, "Optimization under uncertainty: state-of-the-art and opportunities," *Computers & Chemical Engineering*, vol. 28, no. 6-7, pp. 971–983, 2004.
 - [184] N. Sandell, P. Varaiya, M. Athans, and M. Safonov, "Survey of Decentralized Control Methods for Large Scale Systems," *IEEE Transactions on Automatic Control*, vol. 23, no. 2, pp. 108–128, Apr. 1978.
 - [185] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition Structure Generation with Worst Case Guarantees," *Artificial Intelligence*, vol. 111, no. 1-2, pp. 209–238, 1999.
 - [186] A. Schiendorfer, J.-P. Steghöfer, and W. Reif, "Synthesis and Abstraction of Constraint Models for Hierarchical Resource Allocation Problems," in *Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART 2014)*, B. Duval, H. J. van den Herik, S. Loiseau, and J. Filipe, Eds., vol. 2. SciTePress, 2014, pp. 15–27.

- [187] A. Schiendorfer, G. Anders, J.-P. Steghöfer, and W. Reif, “Abstraction of Heterogeneous Supplier Models in Hierarchical Resource Allocation,” *Transactions on Computational Collective Intelligence XX*, vol. 9420, pp. 23–53, 2015.
- [188] A. Schiendorfer, A. Knapp, J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, “Partial Valuation Structures for Qualitative Soft Constraints,” in *Software, Services, and Systems – Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering*, ser. Lecture Notes in Computer Science, R. D. Nicola and R. Hennicker, Eds., vol. 8950. Springer, 2015, pp. 115–133.
- [189] A. Schiendorfer, C. Lassner, G. Anders, W. Reif, and R. Lienhart, “Active Learning for Efficient Sampling of Control Models of Collectives,” in *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2015)*. IEEE Computer Society, 2015, pp. 51–60.
- [190] M. Schillo, P. Funk, and M. Rovatsos, “Using Trust for Detecting Deceitful Agents in Artificial Societies,” *Applied Artificial Intelligence*, vol. 14, no. 8, pp. 825–848, 2000.
- [191] H. Schmeck, C. Müller-Schloer, E. Cakar, M. Mnif, and U. Richter, “Adaptivity and Self-Organization in Organic Computing Systems,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 5, no. 3, pp. 10:1–10:32, 2010.
- [192] C. Schulz, G. Roder, and M. Kurrat, “Virtual Power Plants with combined heat and power micro-units,” in *Proceedings of the International Conference on Future Power Systems*, Nov. 2005, pp. 5 pp.–5.
- [193] P. Scott, S. Thiébaux, M. van den Briel, and P. V. Hentenryck, “Residential Demand Response under Uncertainty,” in *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP 2013)*, ser. Lecture Notes in Computer Science, C. Schulte, Ed., vol. 8124. Springer, 2013, pp. 645–660.
- [194] C. Seren, “A Hybrid Jumping Particle Swarm Optimization Method for High Dimensional Unconstrained Discrete Problems,” in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2011)*. IEEE, 2011, pp. 1649–1656.
- [195] G. D. M. Serugendo, M. P. Gleizes, and A. Karageorgos, “Self-organization in multi-agent systems,” *The Knowledge Engineering Review*, vol. 20, no. 2, pp. 165–189, 2005.
- [196] C. R. Shalizi, K. L. Shalizi, and R. Haslinger, “Quantifying Self-Organization with Optimal Predictors,” *Physical Review Letters*, vol. 93, no. 11, p. 118701, 2004.
- [197] A. Shapiro, “On Complexity of Multistage Stochastic Programs,” *Operations Research Letters*, vol. 34, no. 1, pp. 1–8, 2006.
- [198] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming – Modeling and Theory, Second Edition*, ser. MOS-SIAM Series on Optimization. SIAM, 2014, vol. 16.
- [199] O. Shehory and S. Kraus, “Methods for Task Allocation via Agent Coalition Formation,” *Artificial Intelligence*, vol. 101, no. 1-2, pp. 165–200, 1998.
- [200] J. A. Short, D. G. Infield, and L. L. Freris, “Stabilization of Grid Frequency Through Dynamic Demand Control,” *IEEE Transactions on Power Systems*, vol. 22, no. 3, pp. 1284–1293, Aug. 2007.
- [201] D. C. Smathers and A. A. Akhil, “Operating Environment and Functional Requirements for Intelligent Distributed Control in the Electric Power Grid,” Sandia National Laboratory, Albuquerque, Tech. Rep., Apr. 2000.

-
- [202] R. G. Smith, “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver,” *IEEE Transactions on Computers*, vol. 29, no. 12, pp. 1104–1113, 1980.
 - [203] H. Späth, “Anticlustering: Maximizing the variance criterion,” *Control and Cybernetics*, vol. 15, no. 2, pp. 213–218, 1986.
 - [204] J.-P. Steghöfer and W. Reif, “Die Guten, die Bösen und die Vertrauenswürdigen – Vertrauen im Organic Computing,” *Informatik Spektrum*, vol. 35, no. 2, pp. 119–131, 2012, in German.
 - [205] J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, “A System of Systems Approach to the Evolutionary Transformation of Power Management Systems,” in *Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt*, ser. LNI, M. Horbach, Ed., vol. 220. GI, 2013, pp. 1500–1515.
 - [206] J.-P. Steghöfer, “Large-Scale Open Self-Organising Systems: Managing Complexity with Hierarchies, Monitoring, Adaptation, and Principled Design,” Ph.D. dissertation, University of Augsburg, Germany, 2014.
 - [207] J.-P. Steghöfer, P. Behrmann, G. Anders, F. Siefert, and W. Reif, “HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems,” in *Proceedings of the Ninth International Conference on Autonomic and Autonomous Systems (ICAS)*. IARIA, 2013, pp. 71–76.
 - [208] P. Ströhle, E. H. Gerding, M. de Weerd, S. Stein, and V. Robu, “Online Mechanism Design for Scheduling Non-Preemptive Jobs Under Uncertain Supply and Demand,” in *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS ’14)*, A. L. C. Bazzan, M. N. Huhns, A. Lomuscio, and P. Scerri, Eds. IFAAMAS/ACM, 2014, pp. 437–444.
 - [209] G. Tan, S. A. Jarvis, X. Chen, and D. P. Spooner, “Performance Analysis and Improvement of Overlay Construction for Peer-to-Peer Live Streaming,” *Simulation*, vol. 82, no. 2, pp. 93–106, 2006.
 - [210] S. Tomforde and C. Müller-Schloer, “Incremental Design of Adaptive Systems,” *Journal of Ambient Intelligence and Smart Environments (JAISE)*, vol. 6, no. 2, pp. 179–198, 2014.
 - [211] S. Tomforde, J. Hähner, H. Seebach, W. Reif, B. Sick, A. Wacker, and I. Scholtes, “Engineering and Mastering Interwoven Systems,” in *ARCS 2014 – 27th International Conference on Architecture of Computing Systems, Workshop Proceedings*, W. Stechele and T. Wild, Eds. VDE Verlag / IEEE Xplore, 2014, pp. 1–8.
 - [212] F. Torrent-Fontbona, A. Pla, and B. López, “A New Perspective of Trust Through Multi-Attribute Auctions,” in *Proceedings of the AAAI Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
 - [213] E. P. K. Tsang, *Foundations of Constraint Satisfaction*, ser. Computation in Cognitive Science. Academic Press, 1993.
 - [214] UCTE, “UCTE Operation Handbook – Policy 1: Load-Frequency Control and Performance,” Union for the Co-ordination of Transmission of Electricity, Tech. Rep. UCTE OH P1, 2009.
 - [215] V. Valev, “Set Partition Principles Revisited,” in *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition (SSPR ’98 and SPR ’98)*, ser. Lecture Notes in Computer Science, A. Amin, D. Dori, P. Pudil, and H. Freeman, Eds., vol. 1451. Springer, 1998, pp. 875–881.
 - [216] M. van den Briel, P. Scott, and S. Thiébaux, “Randomized Load Control: A Simple Distributed Approach for Scheduling Smart Appliances,” in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, F. Rossi, Ed. IJCAI/AAAI, 2013.

- [217] D. W. van der Merwe and A. P. Engelbrecht, "Data Clustering using Particle Swarm Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*. IEEE, 2003, pp. 215–220.
- [218] T. Van Zandt, "Hierarchical Computation of the Resource Allocation Problem," *European Economic Review*, vol. 39, no. 3-4, pp. 700–708, Apr. 1995.
- [219] A. Vasalou and J. Pitt, "Reinventing Forgiveness: A Formal Investigation of Moral Facilitation," in *Proceedings of the Third International Conference on Trust Management (iTrust 2005)*, ser. Lecture Notes in Computer Science, P. Herrmann, V. Issarny, and S. Shiu, Eds., vol. 3477. Springer, 2005, pp. 146–160.
- [220] P. Vytelingum, S. D. Ramchurn, T. Voice, A. Rogers, and N. R. Jennings, "Trading Agents for the Smart Electricity Grid," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, Eds., vol. 1–3. IFAAMAS, 2010, pp. 897–904.
- [221] T. Walsh, "Symmetry in Constraint Optimization," in *Proceedings of the Seventh International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon'07)*, vol. 7, 2007.
- [222] —, "Symmetry Breaking Constraints: Recent Results," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012.
- [223] C. Wang and S. M. Shahidehpour, "Effects of ramp-rate limits on unit commitment and economic dispatch," *IEEE Transactions on Power Systems*, vol. 8, no. 3, pp. 1341–1350, Aug. 1993.
- [224] F. Wang, Y. Xiong, and J. Liu, "mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast," in *Proceedings of the 27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007)*. IEEE Computer Society, 2007.
- [225] J. Wang, M. Shahidehpour, and Z. Li, "Security-Constrained Unit Commitment With Volatile Wind Power Generation," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 1319–1327, Aug. 2008.
- [226] Y. Wang and M. P. Singh, "Formal Trust Model for Multiagent Systems," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, M. M. Veloso, Ed., 2007, pp. 1551–1556.
- [227] H. F. Wedde, "DEZENT – A Cyber-Physical Approach for Providing Affordable Regenerative Electric Energy in the Near Future," in *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2012)*, V. Cortellessa, H. Muccini, and O. Demirörs, Eds. IEEE Computer Society, 2012, pp. 241–249.
- [228] M. J. Wooldridge, *An Introduction to MultiAgent Systems (2. ed.)*. Wiley, 2009.
- [229] J. Xu and W. Li, "Solution of Overlapping Coalition Formation Based on Discrete Particle Swarm Optimization," in *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, Oct. 2008, pp. 1–4.
- [230] C. Yingvivatanapong, W.-J. Lee, and E. Liu, "Multi-Area Power Generation Dispatch in Competitive Markets," *IEEE Transactions on Power Systems*, vol. 23, no. 1, pp. 196–203, 2008.
- [231] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 5, pp. 673–685, 1998.
- [232] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 4, pp. 366–379, 2004.

- [233] H. Yu, Z. Shen, C. Leung, C. Miao, and V. R. Lesser, “A Survey of Multi-Agent Trust Management Systems,” *IEEE Access*, vol. 1, pp. 35–50, 2013.
- [234] A. Zafra-Cabeza, M. A. Ridao, I. Alvarado, and E. F. Camacho, “Applying Risk Management to Combined Heat and Power Plants,” *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 938–945, Aug. 2008.
- [235] B. Zhang, P. Luh, E. Litvinov, T. Zheng, F. Zhao, J. Zhao, and C. Wang, “Electricity Auctions with Intermittent Wind Generation,” in *Power and Energy Society General Meeting*. IEEE, July 2011, pp. 1–8.
- [236] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: A New Data Clustering Algorithm and Its Applications,” *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

Own Peer-Reviewed Publications

- G. Anders, F. Siefert, J.-P. Steghöfer, H. Seebach, F. Nafz, and W. Reif, “Structuring and Controlling Distributed Power Sources by Autonomous Virtual Power Plants,” in *Proceedings of the Power & Energy Student Summit 2010 (PESS 2010)*, Oct. 2010, pp. 40–42.
- G. Anders, H. Seebach, F. Nafz, J.-P. Steghöfer, and W. Reif, “Decentralized Reconfiguration for Self-Organizing Resource-Flow Systems Based on Local Knowledge,” in *Proceedings of the 8th IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe 2011)*, Apr. 2011, pp. 20–31.
- G. Anders, J.-P. Steghöfer, F. Siefert, and W. Reif, “Patterns to Measure and Utilize Trust in Multi-agent Systems,” in *Proceedings of the Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems, SASO Workshops (SASOW 2011)*. IEEE Computer Society, 2011, pp. 35–40.
- G. Anders, C. Hinrichs, F. Siefert, P. Behrmann, W. Reif, and M. Sonnenschein, “On the Influence of Inter-Agent Variation on Multi-Agent Algorithms Solving a Dynamic Task Allocation Problem under Uncertainty,” in *Proceedings of the Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)*. IEEE Computer Society, 2012, pp. 29–38.
- G. Anders, F. Siefert, J.-P. Steghöfer, and W. Reif, “A Decentralized Multi-agent Algorithm for the Set Partitioning Problem,” in *Proceedings of the 15th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2012)*, ser. Lecture Notes in Computer Science, I. Rahwan, W. Wobcke, S. Sen, and T. Sugawara, Eds., vol. 7455. Springer, 2012, pp. 107–121.
- G. Anders, J.-P. Steghöfer, F. Siefert, and W. Reif, “A Trust- and Cooperation-Based Solution of a Dynamic Resource Allocation Problem,” in *Proceedings of the 7th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2013)*. IEEE Computer Society, 2013, pp. 1–10.
- G. Anders, A. Schiendorfer, J.-P. Steghöfer, and W. Reif, “Robust Scheduling in a Self-Organizing Hierarchy of Autonomous Virtual Power Plants,” in *27th International Conference on Architecture of Computing Systems, Workshop Proceedings (ARCS 2014)*, W. Stechele and T. Wild, Eds. VDE Verlag / IEEE Xplore, 2014, pp. 1–8.
- G. Anders, F. Siefert, M. Mair, and W. Reif, “Proactive Guidance for Dynamic and Cooperative Resource Allocation under Uncertainties,” in *Proceedings of the Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2014)*. IEEE Computer Society, 2014, pp. 21–30.
- G. Anders, F. Siefert, J.-P. Steghöfer, and W. Reif, “Trust-Based Scenarios – Predicting Future Agent Behavior in Open Self-organizing Systems,” in *Self-Organizing Systems – 7th IFIP TC 6 International Workshop (IWSOS 2013)*, ser. Lecture Notes in Computer Science, W. Elmenreich, F. Dressler, and V. Loreto, Eds., vol. 8221. Springer, 2014, pp. 90–102.
- G. Anders, A. Schiendorfer, F. Siefert, J.-P. Steghöfer, and W. Reif, “Cooperative Resource Allocation in Open Systems of Systems,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 2, pp. 11:1–11:44, 2015.

- G. Anders, F. Siefert, and W. Reif, “A Particle Swarm Optimizer for Solving the Set Partitioning Problem in the Presence of Partitioning Constraints,” in *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2015)*, S. Loiseau, J. Filipe, B. Duval, and H. J. van den Herik, Eds., vol. 2. SciTePress, 2015, pp. 151–163.
- , “A Heuristic for Constrained Set Partitioning in the Light of Heterogeneous Objectives,” in *Agents and Artificial Intelligence: 7th International Conference, ICAART 2015, Lisbon, Portugal, January 10-12, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, B. Duval, H. J. van den Herik, S. Loiseau, and J. Filipe, Eds. Springer, 2015, vol. 9494, pp. 223–244.
- G. Anders, H. Seebach, J.-P. Steghöfer, W. Reif, E. André, J. Hähner, C. Müller-Schloer, and T. Ungerer, “The Social Concept of Trust as Enabler for Robustness in Open Self-Organising Systems,” in *Trustworthy Open Self-Organising Systems*, ser. Autonomic Systems, W. Reif, G. Anders, H. Seebach, J.-P. Steghöfer, E. André, J. Hähner, C. Müller-Schloer, and T. Ungerer, Eds. Springer International Publishing, 2016, pp. 1–16.
- G. Anders, F. Siefert, A. Schiendorfer, H. Seebach, J.-P. Steghöfer, B. Eberhardinger, O. Kosak, and W. Reif, “Specification and Design of Trust-Based Open Self-Organising Systems,” in *Trustworthy Open Self-Organising Systems*, ser. Autonomic Systems, W. Reif, G. Anders, H. Seebach, J.-P. Steghöfer, E. André, J. Hähner, C. Müller-Schloer, and T. Ungerer, Eds. Springer International Publishing, 2016, pp. 17–54.
- B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, and W. Reif, “A Framework for Testing Self-organisation Algorithms,” in *Softwaretechnik-Trends 35*, vol. 1, 2015.
- , “A Research Overview and Evaluation of Performance Metrics for Self-Organization Algorithms,” in *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO Workshops (SASOW 2015)*. IEEE Computer Society, 2015, pp. 122–127.
- B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, A. Knapp, and W. Reif, “An Approach for Isolated Testing of Self-Organization Algorithms,” in *Software Engineering for Self-Adaptive Systems 3*, ser. Lecture Notes in Computer Science, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Springer, 2017, vol. 9640.
- R. Kiefhaber, G. Anders, F. Siefert, T. Ungerer, and W. Reif, “Confidence as a Means to Assess the Accuracy of Trust Values,” in *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012)*, G. Min, Y. Wu, L. C. Liu, X. Jin, S. A. Jarvis, and A. Y. Al-Dubai, Eds. IEEE Computer Society, 2012, pp. 690–697.
- L. Klejnowski, Y. Bernard, G. Anders, C. Müller-Schloer, and W. Reif, “Trusted Community – A Trust-based Multi-Agent Organisation for Open Systems,” in *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*, J. Filipe and A. L. N. Fred, Eds., vol. 1. SciTePress, 2013, pp. 312–317.
- O. Kosak, G. Anders, F. Siefert, and W. Reif, “An Approach to Robust Resource Allocation in Large-Scale Systems of Systems,” in *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2015)*. IEEE Computer Society, 2015, pp. 1–10.
- F. Nafz, H. Seebach, J.-P. Steghöfer, G. Anders, and W. Reif, “Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach,” in *Organic Computing – A Paradigm Shift for Complex Systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer, 2011, pp. 79–93.
- A. Schiendorfer, G. Anders, J.-P. Steghöfer, and W. Reif, “Abstraction of Heterogeneous Supplier Models in Hierarchical Resource Allocation,” *Transactions on Computational Collective Intelligence XX*, vol. 9420, pp. 23–53, 2015.
- A. Schiendorfer, A. Knapp, J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, “Partial Valuation Structures for Qualitative Soft Constraints,” in *Software, Services, and Systems – Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software*

-
- Engineering*, ser. Lecture Notes in Computer Science, R. D. Nicola and R. Hennicker, Eds., vol. 8950. Springer, 2015, pp. 115–133.
- A. Schiendorfer, C. Lassner, G. Anders, R. Lienhart, and W. Reif, “Active Learning for Abstract Models of Collectives,” in *Proceedings of the 3rd International Workshop on “Self-optimisation in Organic and Autonomic Computing Systems” (SAOS 2015)*, Mar. 2015.
- A. Schiendorfer, C. Lassner, G. Anders, W. Reif, and R. Lienhart, “Active Learning for Efficient Sampling of Control Models of Collectives,” in *Proceedings of the 9th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2015)*. IEEE Computer Society, 2015, pp. 51–60.
- F. Siefert, G. Anders, M. Sommer, and W. Reif, “A Generic Framework for Simulating the EEX Power Market in Agent-Based Energy Management Applications,” in *Tagungsband des Power and Energy Student Summit 2012*, Jan. 2012.
- J.-P. Steghöfer, G. Anders, F. Siefert, and W. Reif, “A System of Systems Approach to the Evolutionary Transformation of Power Management Systems,” in *Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt*, ser. LNI, M. Horbach, Ed., vol. 220. GI, 2013, pp. 1500–1515.
- J.-P. Steghöfer, G. Anders, J. Kantert, C. Müller-Schloer, and W. Reif, “An Effective Implementation of Norms in Trust-Aware Open Self-Organising Systems,” in *Proceedings of the Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO Workshops (SASOW 2014)*. IEEE Computer Society, 2014, pp. 76–77.
- J.-P. Steghöfer, P. Behrmann, G. Anders, F. Siefert, and W. Reif, “HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems,” in *Proceedings of the Ninth International Conference on Autonomic and Autonomous Systems (ICAS)*. IARIA, 2013, pp. 71–76.